175 PTAS
64
mi computer
CURSO PRACTICO DEL ORDENADOR PERSONAL,
EL MICRO Y EL MINIORDENADOR
ZX Spectrum
Editorial Delta S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen VI-Fascículo 64

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, 08008 Barcelona
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-034-7 (tomo 6)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 038504
Impreso en España-Printed in Spain-Marzo 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S.A. (Paseo de Gracia, 88, 5.°, 08008 Barcelona), o también con talón bancario remitido a la misma dirección.

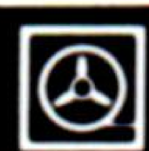
c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

No se efectúan envíos contra reembolso.





Conocer el entorno

¿Cómo combinar los "sentidos" del robot estudiados hasta ahora? Es el tema que trataremos en este capítulo

Recorrer la sala
El "camarero" hace un recorrido aleatorio desde la barra hasta un punto aproximadamente opuesto, describiendo una trayectoria por lo general en diagonal. Se desplaza lentamente, combinando entradas de los sensores visuales y de proximidad, para no chocar contra las mesas o las personas

Llenar las copas
El "camarero" deja la bandeja vacía sobre la barra y encuentra otra con copas vacías y limpias esperándolo. Las llena utilizando una entrada visual para posicionar una boquilla en cada copa, vertiendo luego una cantidad preestablecida de líquido

Apreciar la diferencia
El camarero-robot está programado para distinguir entre una copa llena y una vacía. Sintetizando lo que ve en la bandeja y las diferencias discretas de masa, el robot determina en qué momento debe "repostar"

Tomar pedidos
El camarero-robot puede reconocer varias instrucciones habladas sencillas. "Oye" a la gente que lo llama y adapta su recorrido orientándose en la dirección de la instrucción

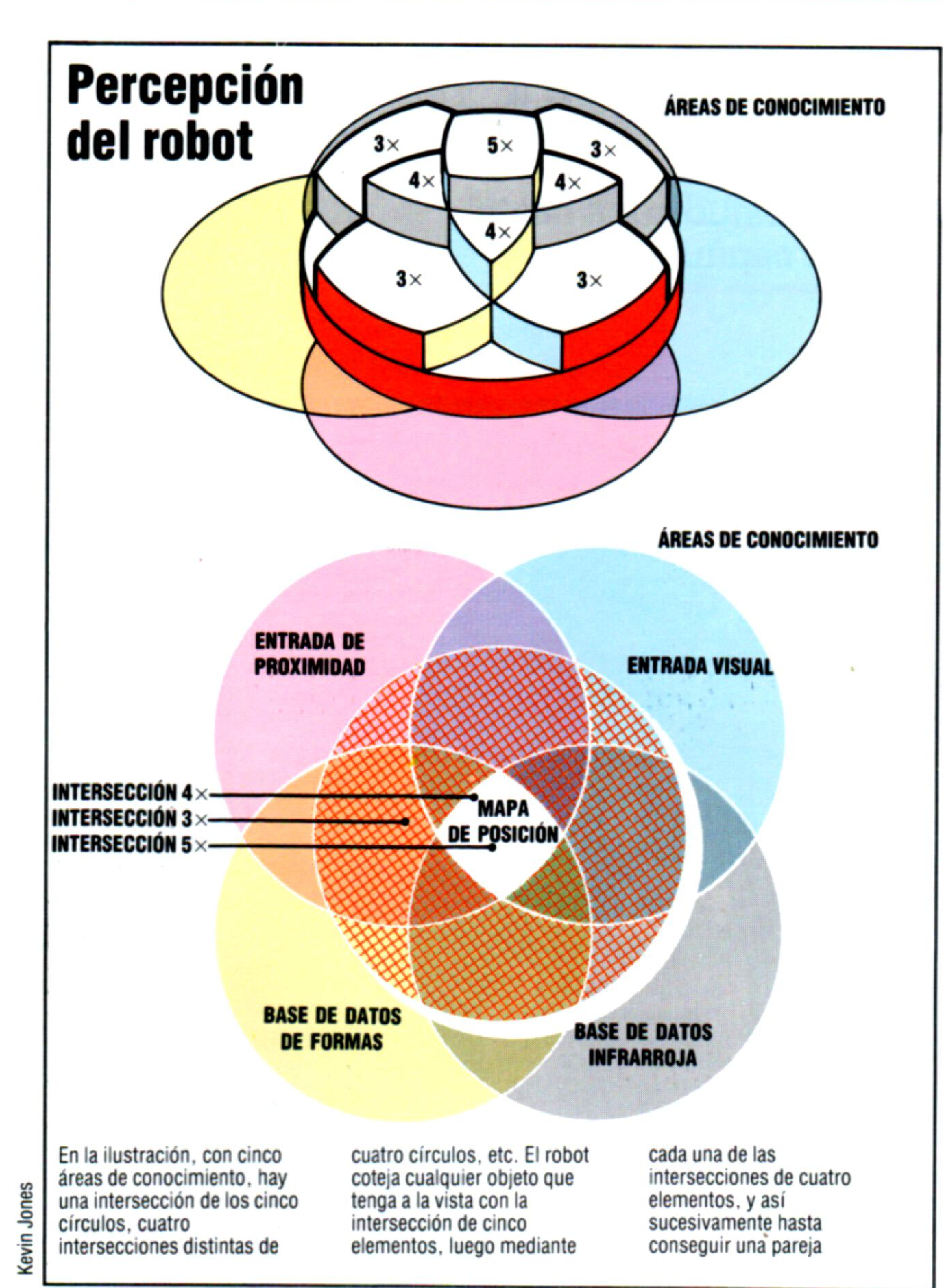
Mike Brownlow

Cuando examinábamos los sensores que puede utilizar un robot para obtener cierto conocimiento del mundo en el cual se mueve, consideramos cada tipo de entrada sensorial (vista, sonido, tacto) como si se la empleara de forma aislada. Éste sería un supuesto bastante factible si el robot tuviera sólo un sensor; pero, en la práctica, los más perfeccionados disponen de varios. Para comprender su entorno el robot ha de ser capaz de integrar estas entradas sensoriales empleando cada una de ellas como comprobación de las otras, con el objeto de construirse un modelo interno del mundo circundante.

Este hecho no debería sorprender, porque así es, al parecer, cómo actúan también los seres humanos. Nuestros sentidos no existen aisladamente: estamos utilizando de manera constante la entrada de un sentido como verificación de la entrada de otro, y la consecuencia es la construcción de una imagen muy completa de nuestro entorno. El más claro ejemplo de ello nos lo proporcionan los estudios realizados en personas ciegas de nacimiento a las que se ha dotado de vista gracias a una intervención quirúrgica. Estos pacientes a menudo sorprenden a sus cirujanos por la velocidad con la cual pueden hacer un uso completo de su vista. Ello se debe a que los ciegos poseen un conocimiento muy exacto del mundo que les rodea como consecuencia de poder tocar objetos, desplazarse por él y escuchar descripciones del entorno. Por consiguiente, una vez dotados de visión, pueden utilizar este conocimiento y aplicarlo para interpretar lo que perciben sus ojos.

Para poder obtener los mejores resultados con los robots, debemos posibilitar la interacción de sus sentidos como si se tratara de los de una persona. Por ejemplo, un robot diseñado para asir objetos podría ser capaz de cogerlos "a ciegas", pero será mucho mejor que disponga de un sistema visual, porque entonces podrá localizar los objetos aun cuando estén ligeramente desplazados de sitio o en un ángulo diferente a aquel en el que el robot espera hallarlos en función de su programación. Para hacer esto, el robot debe construirse alguna clase de modelo interno de su entorno utilizando las entradas de todos sus sensores. Debe ser capaz de mirar el objeto y reconocerlo, debe posicionar luego su efector final y efectuar los cálculos necesarios para levantar el objeto.

A sus órdenes...
Hemos creado un camarero-robot imaginario que debe sintetizar diversas entradas sensoriales para desempeñar diligentemente su función. Su mayor problema reside en el hecho de que los invitados están moviéndose de manera constante, lo que significa que debe actualizar su modelo interno del espacio tomando en consideración esta circunstancia



En la ilustración, con cinco áreas de conocimiento, hay una intersección de los cinco círculos, cuatro intersecciones distintas de cuatro círculos, etc. El robot coteja cualquier objeto que tenga a la vista con la intersección de cinco elementos, luego mediante cada una de las intersecciones de cuatro elementos, y así sucesivamente hasta conseguir una pareja

Kevin Jones

Interface de intersección
Normalmente los robots tienen varias fuentes de conocimiento a su disposición: algunas son sus bases de datos preprogramados (de siluetas de objetos comunes y "firmas" infrarrojas, p. ej.), otras son bases de datos empíricas (como el mapa corriente de su entorno que posee el robot) y unas terceras son canales de entradas sensoriales (como proximidad y forma de objetos). En el caso ideal (cuando el robot "conoce" su posición y "comprende" su mundo circundante), la intersección de estas áreas de conocimiento debe identificar con carácter exclusivo cualquier objeto que haya a la vista del robot

La ilustración más simple de este modelo interno es la del robot que salva laberintos (véase p. 1252), que utiliza sensores para calcular la posición de las paredes del laberinto, construyendo un mapa interno bidimensional a medida que va avanzando. Si ampliamos esta idea a un brazo, el mapa debería ser tridimensional. Al dotar de vista al robot el mapa tridimensional adquiere inmediatamente colores, variaciones de brillo y patrones que ningún sensor táctil puede detectar. Con cierta medida de reconocimiento de habla, el robot puede agregar información hablada a su modelo del mundo circundante.

Uno de los problemas con los que se enfrentan los diseñadores cibernéticos que están tratando de posibilitar que el robot interprete su entorno es que el mundo no es estático y cambia de forma constantemente. Por consiguiente, es necesario que el robot, asimismo, esté equipado con algún medio de dar cabida a tales cambios.

Si consideramos un robot que esté programado para llevar a cabo alguna tarea sencilla, como apilar ladrillos, se hace evidente la magnitud de este problema. Si los ladrillos son de distinto tamaño se los debe colocar uno encima de otro muy cuidadosamente, y si el centro de gravedad de cada uno de ellos se desplaza fuera de la superficie de la base de los ladrillos, toda la pila se derrumbará. Pero ¿qué puede saber un robot acerca de las leyes de la gravedad? Y si la pila se viene abajo, ¿comprenderá lo que ha sucedido y emprenderá la acción necesaria?

Resolución de problemas

Para abordar este problema existen dos enfoques principales. El primero es programar el robot con datos que incluyan un curso de acción prescrito para cualquier eventualidad. Esto obviamente limitará el entendimiento del robot a ciertas tareas claramente definidas. El robot apilador de ladrillos se programará, entonces, con instrucciones para asegurar que cada ladrillo se coloque exactamente encima del que se encuentra debajo, con el centro de gravedad de uno situado directamente encima del centro de gravedad de otro.

El segundo enfoque es el que defienden quienes sostienen que la única forma de que un robot pueda llegar alguna vez a comprender su entorno es aprendiéndolo por sí mismo. Éste es un campo de la ciencia informática que se denomina aprendizaje o *heurística*. Según este enfoque, el robot se programa para llevar a cabo una tarea determinada y se le proporciona realimentación, ya sea proveniente de una persona o bien de su propio sensor, que le dice lo bien que lo ha hecho. En relación a esta realimentación el robot modificará su propio programa interno (su propio modelo del mundo) con el fin de mejorar su rendimiento y construirse una "biblioteca de referencia" que le ayudará a hacer frente a futuras tareas. El robot que salva laberintos actúa de esta forma cuando intenta hallar la mejor ruta a través de un laberinto. Utilizando sus sensores puede detectar cualquier callejón sin salida, emprendiendo la acción necesaria para desandar el camino y volver a intentarlo con uno nuevo. Lamentablemente, no existe ningún programa de aprendizaje que se pueda utilizar cada vez que un robot necesita aprender una tarea.

Después de que el robot ha "aprendido" la lección correspondiente, ha de almacenarla, sea cual fuere, en forma de programa para ordenador. Esta tarea se conoce como *representación del conocimiento*. Tradicionalmente, el conocimiento del robot se puede almacenar como líneas de código similares a las de un programa común para ordenador. Pero las técnicas de la inteligencia artificial han llevado a otros enfoques. Existe en uso una amplia gama de técnicas, pero las más comunes incluyen *reglas de producción, redes semánticas* y *marcos*.

Las reglas de producción responden a la forma de las construcciones IF...THEN y son simples sentencias de la realidad. Por consiguiente, un robot podría almacenar su conocimiento en la forma: SI (IF) hay una pared de ladrillos delante de ti, ENTONCES (THEN) no puedes avanzar. Puede haber toda una secuencia de reglas de este tipo; ofrecen la ventaja de ser fáciles de escribir y fáciles de comprender para el programador que esté escribiendo el programa. Pero existe el pequeño problema de que el robot también tiene que comprenderlas: necesita lo que suele llamarse un *motor de inferencias* para ser capaz de interpretar estas reglas como un curso de acción. Los programas que utilizan reglas de producción se pueden escribir en un lenguaje convencional, como el BASIC, pero lo más común es



que se escriban en un lenguaje *declarativo*, como el PROLOG, que está mejor diseñado para tratar esta clase de conocimiento. Ello se debe a que, a diferencia de los tradicionales, los lenguajes declarativos no ejecutan sus instrucciones de una en una. En cambio el programa busca continuamente un conjunto dado de circunstancias al cual se puedan aplicar una o más reglas. Cuando esto sucede, esa regla determinada "se dispara" y es ejecutada, lo que, a su vez, puede conducir a que se "disparen" otras reglas.

Las redes semánticas son una forma de estructuras gráficas que se utilizan para representar el conocimiento, y se puede pensar en ellas como una red de relaciones entre diversos elementos del conocimiento. La razón por la cual se denominan redes semánticas, en vez de simplemente redes o gráficos, reside en que los enlaces individuales pueden poseer algún significado en sí mismos. Un arco que enlace dos nudos puede ser un arco que indique la existencia de una clase especial de relación entre aquellos dos nudos. Por lo tanto, un nudo etiquetado "mesa" podría estar unido a un nudo etiquetado "mueble". En este ejemplo, la relación es que una mesa es un tipo de mueble; de modo que podemos decir que el enlace es un enlace de "tipo".

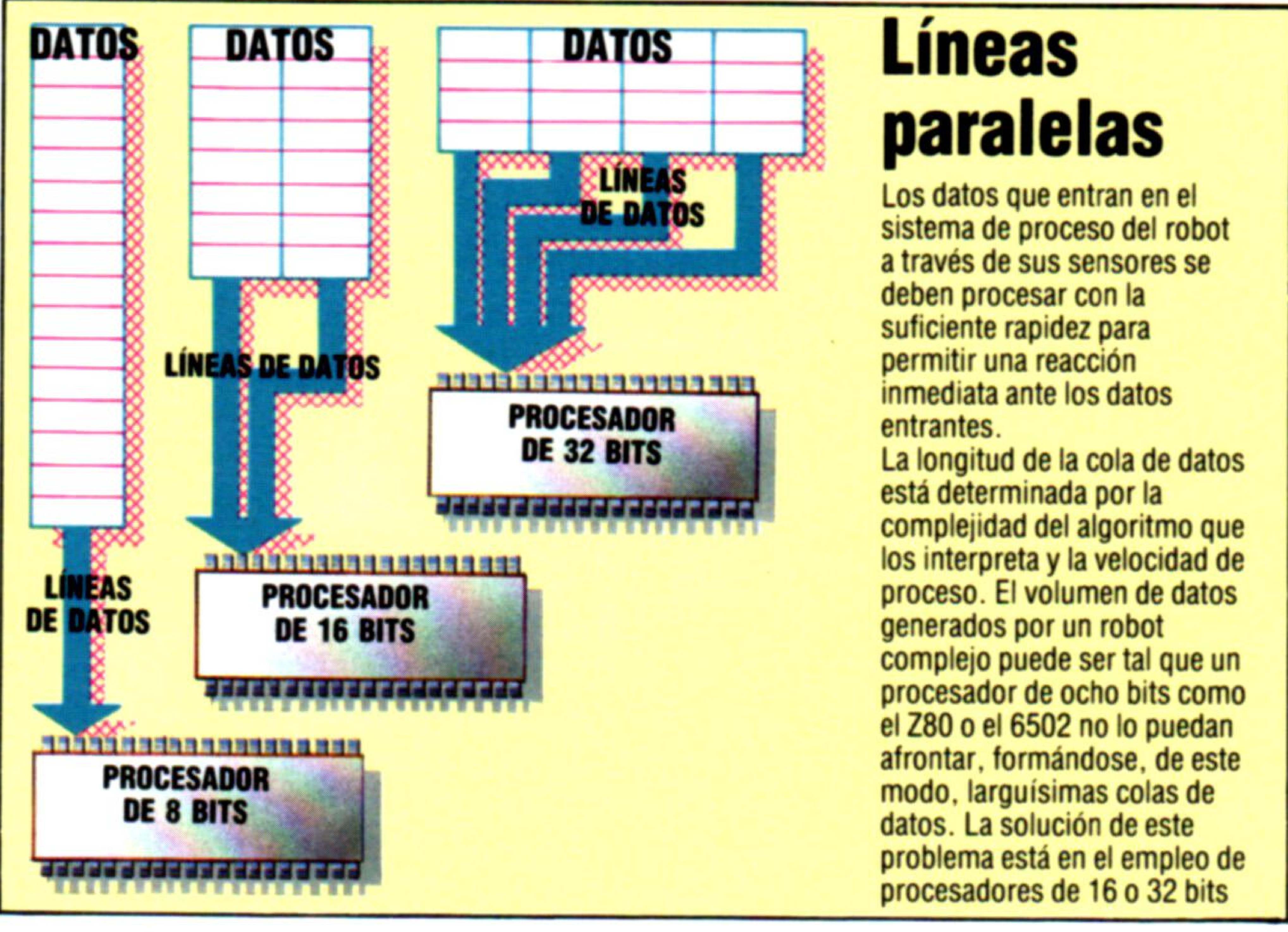
Esta clase de representación del conocimiento se puede programar en un lenguaje convencional. Usted puede intentarlo empleando el BASIC y representando los diversos nudos y enlaces mediante variables en serie. Pero lo más común es utilizar uno de los lenguajes de la inteligencia artificial, como el LISP, porque hacen que la expresión de estas complejas relaciones con nombre sea mucho más sencilla.

Los marcos son una especie de cuestionario en blanco que se diseña específicamente para cada tipo de situación con la que se pueda enfrentar un robot. La idea es muy fácil de comprender y se podría programar de inmediato en BASIC utilizando simples matrices en serie bidimensionales: una dimensión para la "pregunta" y otra para las "respuestas".

En este enfoque se considera que el robot no posee un conocimiento completo acerca de una situación hasta haber rellenado todos los puntos del cuestionario. Sólo entonces podrá emprender la acción apropiada. Un refinamiento de este método consiste en poner a disposición del robot una gran selección de marcos. Una de las tareas del mismo será, entonces, seleccionar el marco adecuado para una situación dada.

Un aspecto importante de la representación del conocimiento, del cual hemos hablado muy brevemente, es el papel que representan los lenguajes de programación. El LISP, por ejemplo, al ser un lenguaje de proceso de listas, es especialmente adecuado para esta clase de enfoque para el almacenamiento y la recuperación de datos. La elección de lenguaje, en consecuencia, simplificará la representación del conocimiento de determinadas formas.

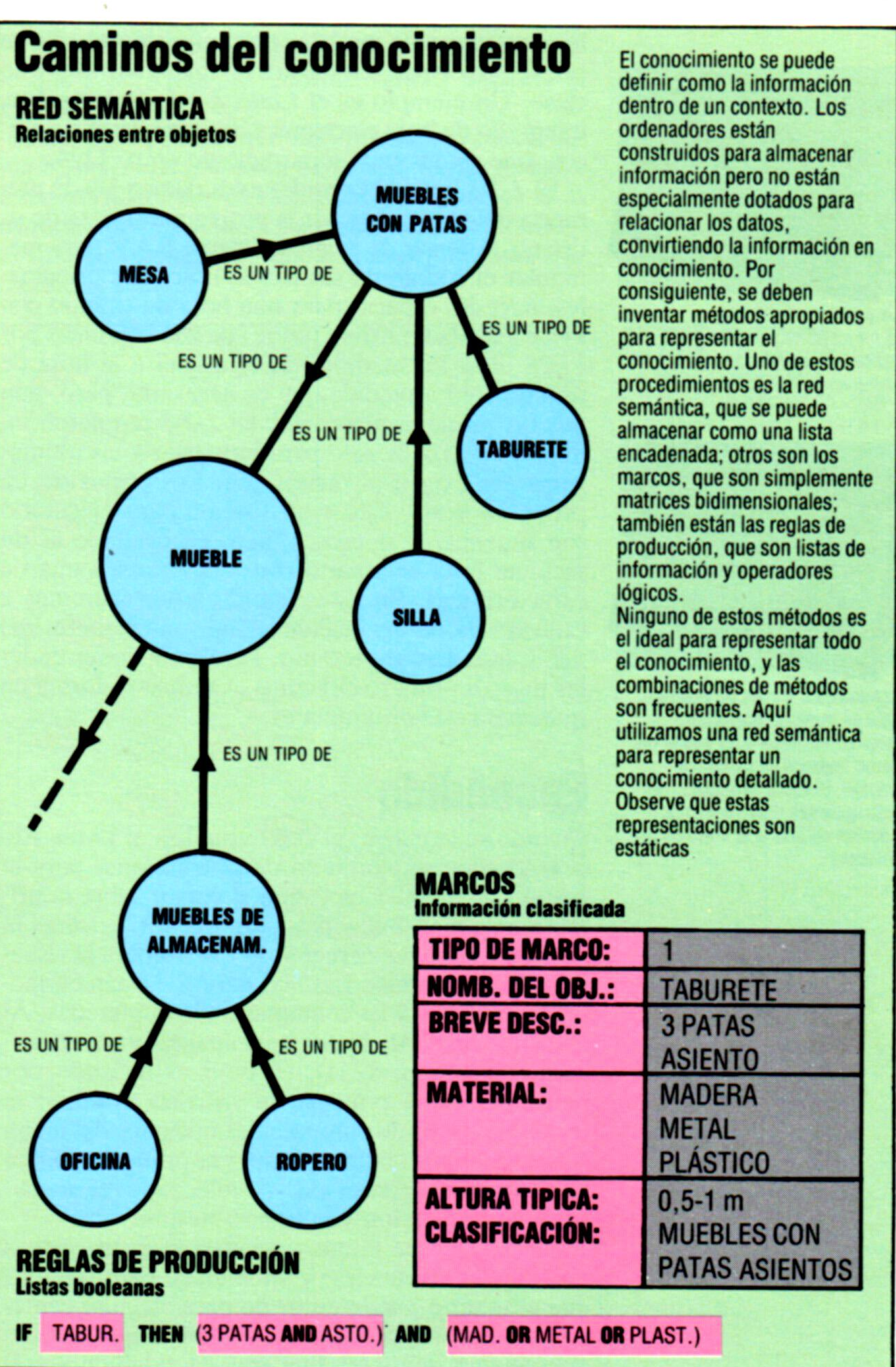
En este capítulo hemos analizado unos cuantos métodos para programar al robot con un entendimiento más completo de su entorno. Este tema todavía sigue siendo objeto de considerable investigación en los departamentos de ciencias de universidades de todo el mundo y los elementos clave son la utilización de sensores, la realimentación, el aprendizaje y la representación del conocimiento.



Líneas paralelas

Los datos que entran en el sistema de proceso del robot a través de sus sensores se deben procesar con la suficiente rapidez para permitir una reacción inmediata ante los datos entrantes.
La longitud de la cola de datos está determinada por la complejidad del algoritmo que los interpreta y la velocidad de proceso. El volumen de datos generados por un robot complejo puede ser tal que un procesador de ocho bits como el Z80 o el 6502 no lo puedan afrontar, formándose, de este modo, larguísimas colas de datos. La solución de este problema está en el empleo de procesadores de 16 o 32 bits

Caminos del conocimiento



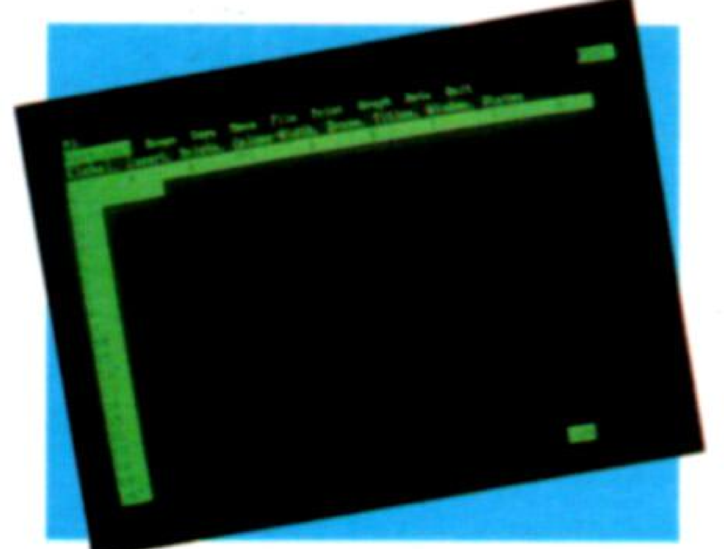
TIPO DE MARCO:	1
NOMB. DEL OBJ.:	TABURETE
BREVE DESC.:	3 PATAS ASIENTO
MATERIAL:	MADERA METAL PLÁSTICO
ALTURA TIPICA: CLASIFICACIÓN:	0,5-1 m MUEBLES CON PATAS ASIENTOS

El conocimiento se puede definir como la información dentro de un contexto. Los ordenadores están construidos para almacenar información pero no están especialmente dotados para relacionar los datos, convirtiendo la información en conocimiento. Por consiguiente, se deben inventar métodos apropiados para representar el conocimiento. Uno de estos procedimientos es la red semántica, que se puede almacenar como una lista encadenada; otros son los marcos, que son simplemente matrices bidimensionales; también están las reglas de producción, que son listas de información y operadores lógicos.
Ninguno de estos métodos es el ideal para representar todo el conocimiento, y las combinaciones de métodos son frecuentes. Aquí utilizamos una red semántica para representar un conocimiento detallado. Observe que estas representaciones son estáticas

Gran macro

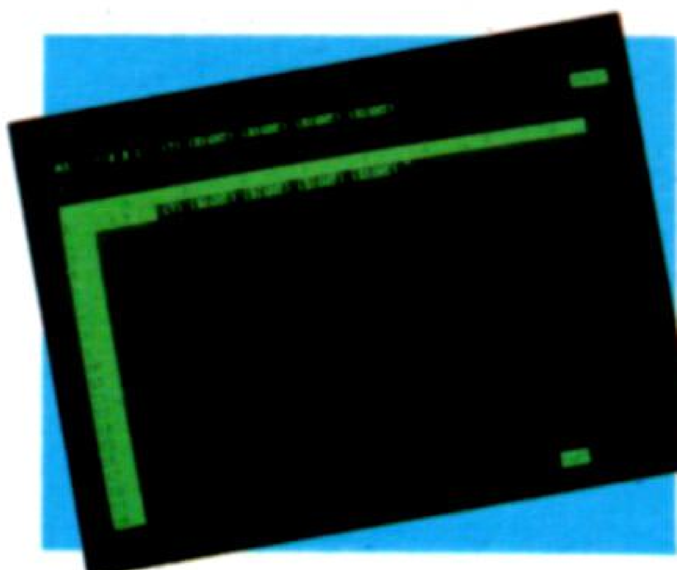
Examinemos el uso de macro de teclado y otras características en el paquete «1-2-3», producido por Lotus

Los programas de hoja electrónica que hemos examinado hasta ahora han sido diseñados para micros personales como el BBC, el Spectrum, el Commodore 64 y el Sinclair QL. Estos programas se ven necesariamente entorpecidos por la limitada cantidad de RAM disponible para el programa en sí y para los modelos desarrollados. Programas similares escritos para el IBM PC, el ACT Apricot y otras máquinas de oficina pueden sacar partido de grandes memorias residentes y una mayor velocidad de proceso. A consecuencia de ello, algunos de los programas de modelos más innovadores sólo se pueden ejecutar en costosas máquinas de esta clase. Un ejemplo es el *Lotus 1-2-3*, un programa integrado de hoja electrónica, base de datos y gráficos, que estudiamos someramente en p. 1124.



La pantalla del *1-2-3*
El menú principal se trae a la pantalla pulsando la tecla /, como en el *VisiCalc*

El *1-2-3* impone considerables demandas de memoria del ordenador por la propia naturaleza de su diseño. Además de exigir suficiente RAM para manipular el código de sus tres aplicaciones principales, necesita espacio para una hoja de trabajo con una capacidad máxima teórica de 256 columnas por 1 028 filas. La memoria se le asigna a la hoja de trabajo sólo a medida que es necesaria; pero, aun así, las primeras versiones del *1-2-3* requieren un mínimo de 128 K sólo para funcionar y las últimas versiones exigen al menos 256. Los costos de un programa como éste y un sistema para ejecutarlo son sumamente onerosos, pero el resultado es un paquete para el usuario con una amplia gama de características. En este capítulo le enseñaremos a utilizar una de las facilidades más interesantes del *1-2-3*, la macro de teclado. Pero para comprender las macros primero debemos examinar la forma en que opera este programa.



Economía a través de macros
Ésta es nuestra macro para asignar nombres a las zonas tal como aparece en la hoja de trabajo. Si fuera necesario, las instrucciones de la macro se podrían colocar a lo largo de una columna

Encendido

Cuando se lo activa, el *1-2-3* visualiza el Lotus Access System, un conjunto de instrucciones para la gestión de datos. Colocando el cursor sobre la primera opción, 1-2-3, y pulsando Return, se carga la hoja de trabajo en la memoria y se prepara la visualización en pantalla. Las filas del *1-2-3* están etiquetadas con letras y las columnas están numeradas. Al igual que en el *Multiplan*, que analizamos anteriormente (véase p. 1244), el *1-2-3* es activado por menú. El menú principal se visualiza al pulsar la tecla /. A partir de entonces, las opciones del menú se seleccionan o bien digitando la primera letra de la instrucción apropiada, o colocando el cursor sobre la instrucción y pulsando Return.

El *1-2-3* posee tantas opciones de instrucciones que hay varios niveles de submenús. Esto significa que el usuario puede utilizarlo para efectuar centenares de tareas, pero también significa que algunas operaciones requieren una gran cantidad de pulsaciones de teclas (véase diagrama). Para ilustrar este punto, tomemos un ejemplo. El *1-2-3* permite designar a una celda o a una zona de celdas con una etiqueta identificadora. Cuando se desea actuar sobre la zona designada (incluyéndola en una fórmula, p. ej.) se emplea el nombre asignado en lugar de la referencia a la celda, así:

A3−B3=C3	VENTAS−COSTO=BENEFICIO
referencias a celdas	referencias a nombres

La asignación de nombres a las zonas simplifica y acelera la tarea de buscar cosas en una gran hoja de trabajo. Para asignar un nombre a un grupo de cuatro celdas en el *1-2-3*, se requieren las siguientes pulsaciones de tecla:

/ — visualiza el menú principal;
R(ange,serie) — le dice al 1-2-3 que se va a efectuar una operación sobre un pequeño grupo de celdas, en vez de sobre la hoja de trabajo completa;
N(ombre) — a la serie de celdas identificadas se le va a asignar un nombre;
C(rear) — preparar al 1-2-3 para aceptar un nombre y asignarlo;
Digite el nombre a asignar: "COMIENZO", por ejemplo;
Return para aceptar la celda activa como el principio de la serie;
Cursor a la derecha cuatro espacios;
Return para aceptar la celda activa como el final de la serie.

Por consiguiente, este proceso exige 10 pulsaciones de teclas, más el nombre propiamente dicho.

Para reducir este proceso a un número más manejable, el *1-2-3* permite el empleo de macros de teclado. Las macros son como programas simples, escritos en el lenguaje operativo del paquete. Se crean almacenando las pulsaciones de teclas requeridas en una pequeña porción de la hoja de trabajo, asignándole un nombre a la posición, asignando luego el nombre a una tecla específica. A partir de ese momento, el *1-2-3* efectuará las pulsaciones de forma automática cada vez que pulse la tecla asignada juntamente con una de función especial, etiquetada ALT en el IBM PC y en las máquinas compatibles con el mismo.

Para automatizar el proceso de asignación de nombres a celdas, comenzamos por destinar para la macro una selección de celdas de la hoja de trabajo. Estas celdas se deben elegir cuidadosamente por dos motivos. En primer lugar, deben ocupar un espacio a salvo dentro de la hoja de trabajo, una zona en la que jamás se colocarán datos. En segundo lugar, como hemos mencionado, el *1-2-3* sólo destina espacio de memoria a las celdas que se activan. Una celda se activa siempre que sea señalada por el cursor, de modo que a los espacios vacíos entre celdas de datos se les otorgará un espacio de

la memoria. Por consiguiente, si se colocara una macro hacia la derecha, muy alejada del resto de la hoja de trabajo, las celdas vacías podrían devorar varios Kbytes de memoria utilizable. Por este motivo, a menudo es preferible colocar las macros sobre el margen izquierdo de la hoja de trabajo, en las columnas A, B y C, por ejemplo. Si todas las fórmulas de la hoja de trabajo se escriben como para funcionar de izquierda a derecha, las zonas de las macros no se tocarán en absoluto.

Vamos a construir nuestra macro de asignación de nombres en la columna A. Si la cantidad de pulsaciones se vuelve demasiado grande como para caber en una sola celda, se puede colocar a la macro en filas subsiguientes de la misma columna. Señalando el cursor a la celda A1, realizamos las pulsaciones de teclas necesarias:

'/ R N C

En este punto, el *1-2-3* aguarda para aceptar desde el teclado el nombre elegido. Se inserta una pausa entrando un signo de interrogación encerrado entre paréntesis, (?). El programa esperará hasta que el usuario pulse Return para proseguir con la macro. El formato de paréntesis se emplea coherentemente siempre que se desea una acción que no se pueda indicar mediante una tecla específica. En esto están incluidos los movimientos del cursor, que se indican digitando una palabra de dirección entre paréntesis. Los retornos de carro se indican mediante un tilde, ˜. Entonces continúa la macro:

'/ R N C (?)˜(derecha) (derecha) (derecha) (derecha)˜

Ahora hemos entrado todas las pulsaciones de tecla que necesitamos para asignar un nombre a una zona. Éste es el cuerpo de nuestra macro.

El siguiente paso consiste en asignarle un nombre a la zona con la etiqueta de una tecla, como N de nombre. Lamentablemente, aquí nos encontramos con un círculo vicioso. ¡Debemos realizar todas las pulsaciones de teclas que acabamos de entrar en nuestra macro designadora de series, porque aún no le hemos asignado un nombre a nuestra macro ni la hemos almacenado! Colocamos el cursor en la celda A1 y digitamos:

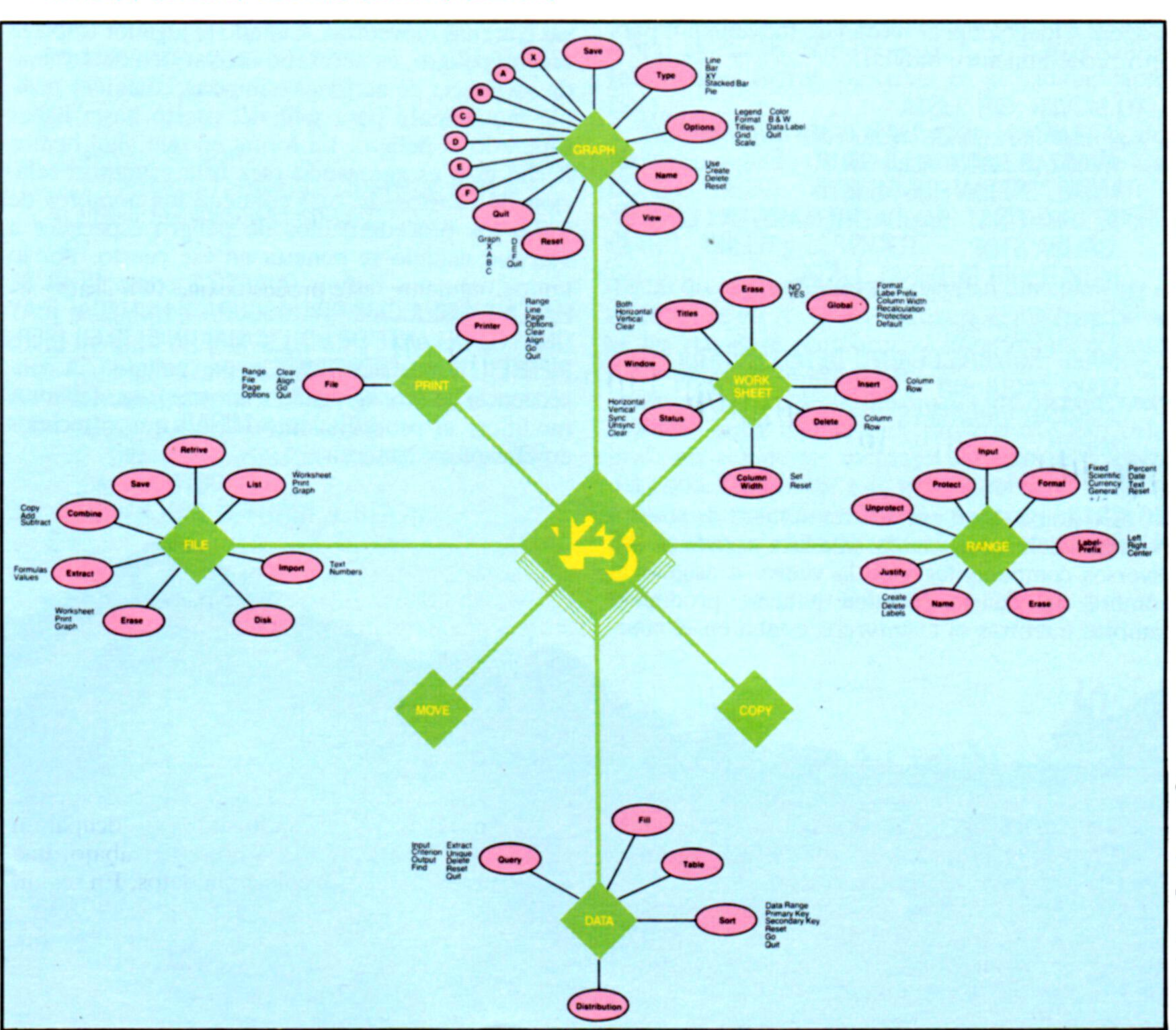
'/ R N C \ N Return Return

El carácter \ se utiliza para indicar que se debe pulsar la tecla ALT; de modo que \N, nombre de nuestra macro, significa para el *1-2-3* ALT N. Ahora que ya se le ha asignado un nombre a la zona, pulsando ALT y N juntas se activará automáticamente la secuencia de pulsaciones de teclas almacenada. A partir de este punto, la asignación de un nombre a una zona se puede efectuar digitando:

ALT–N NOMBRE Return

lo que representa una enorme mejora respecto al proceso original.

Este ejemplo, aunque útil, representa una mínima muestra del potencial de las macros de teclado. Esencialmente no existen límites en cuanto a la cantidad de pulsaciones de teclas ni a la cantidad o el tipo de operaciones que se pueden automatizar mediante macros.



Cortesía de Lotus Development Corp.

Racimo de instrucciones del *1-2-3*
Tal como se refleja en la ilustración, el paquete de Lotus posee varios niveles de menús. Estas instrucciones están incorporadas. Otras las puede definir el usuario a través de las macros de teclado

Conjurar el peligro

Nos corresponde estudiar otra fase de nuestro juego: cómo desplazarse entre escenarios y enfrentarse a los "peligros"

Ahora que ya hemos implementado la estructura básica del juego debemos considerar los procedimientos para desplazarse de un cuarto a otro cuarto. Permitiremos cuatro direcciones de movimiento: norte, sur, este y oeste.

```
TO N
  MOVE "N :LISTA.SALIDAS
END

TO S
  MOVE "S :LISTA.SALIDAS
END

TO E
  MOVE "E :LISTA.SALIDAS
END

TO O
  MOVE "O :LISTA.SALIDAS
END
```

Un procedimiento denominado MOVER verifica primero que el jugador se pueda desplazar en esa dirección, y luego deja el verdadero movimiento para otro procedimiento, MOVER1.

```
TO MOVER :DIR :LISTA
  IF EMPTY? :LISTA THEN PRINT [NO PUEDE
  AVANZAR HACIA ALLI] STOP
  MAKE "SALIDA FIRST :LISTA
  IF :DIR=FIRST :SALIDA THEN MOVER1 LAST
  :SALIDA STOP
  MOVER :DIR BUTFIRST :LISTA
END
  TO MOVER1 :NO
  MAKE :NOMBRE.CUARTO DETALLES.AQUI
  MAKE "AQUI :NO
  ASIGNAR.VARIABLES
  MIRAR
END
```

MOVER1 toma como entrada un número de cuarto. Primero vuelve a ensamblar una lista a partir de sus diversos componentes y se la vuelve a asignar al nombre del cuarto (pueden haberse producido cambios mientras el aventurero estaba en el cuarto). Luego modifica HERE para un nuevo número de cuarto y vuelve a asignar las diversas listas. El procedimiento que emplea es éste:

```
TO DETALLES.AQUI
  OUTPUT (LIST :DESCRIPCION :CONTENIDO
  :LISTA.SALIDAS)
END
```

Éste utiliza la primitiva LIST, que hace una lista de sus entradas. La diferencia entre LIST y SENTENCE se explica mejor a través de un ejemplo:

```
LIST [A] [B] [C] produce [[A] [B] [C]]
SENTENCE [A] [B] [C] produce [A B C]
```

Dado que queremos conservar los componentes individuales en forma de sublistas, aquí necesitamos utilizar LIST en lugar de SENTENCE.

Los peligros del juego

Por lo general, en todo juego de aventuras hay ciertos "peligros" que evitar, como serpientes venenosas o arenas movedizas. Cuando el jugador tropieza con un peligro, es necesario activar una determinada secuencia de acciones e impedir cualquier posible movimiento para salir del cuarto hasta haber superado el peligro. La forma en que aquí hemos hecho esto es agregando otra lista a nuestra relación de cuartos, la cual contiene los nombres de todos los procedimientos de peligro especiales a ejecutar cuando se penetra en ese cuarto. Por lo tanto, debemos definir CUARTO.2 como [[[UD SE HALLA EN UNA CAVERNA OSCURA Y HUMEDA] [HAY UNA LUZ DELANTE DE UD]] [CAJA] [[N 5] [E 6]] [SERPIENTE]], donde SERPIENTE es un "peligro". A consecuencia de este agregado a nuestra lista, debemos modificar el procedimiento MIRAR que ofrecimos en el capítulo anterior:





```
TO MIRAR
  PRINTL :DESCRIPCION
  PRINT "
  PRINT [UD VE:]
  IF EMPTY? :CONTENIDO THEN PRINT [NADA
  ESPECIAL] ELSE PRINT :CONTENIDO
  PRINT "
  PRINT [PUEDE IR:] IMPRIMIR.SALIDAS
  :LISTA.SALIDAS
  PRINT "
  IF PELIGRO? THEN RUN :PELIGROS
END
```

RUN es una primitiva muy poderosa del LOGO. Toma como entrada una lista y ejecuta los procedimientos que contenga esa lista. Aquí, [SERPIENTE] podría ser asociada a PELIGROS, de modo que RUN :PELIGROS ejecutaría SERPIENTE.

```
TO PELIGRO?
  IF EMPTY? :PELIGROS THEN OUTPUT "FALSE
  OUTPUT "TRUE
END
```

Ahora es necesario modificar algunos otros procedimientos para tener en cuenta estos peligros:

```
TO ASIGNAR.VARIABLES
  MAKE "NOMBRE.CUARTO WORD "CUARTO. :AQUI
  MAKE "CUARTO THING :NOMBRE.CUARTO
  MAKE "DESCRIPCION DESCRIPCION :CUARTO
  MAKE "CONTENIDO CONTENIDO :CUARTO
  MAKE "LISTA.SALIDAS LISTA.SALIDAS :CUARTO
  MAKE "PELIGROS PELIGROS :CUARTO
END

TO PELIGROS :CUARTO
  OUTPUT ITEM 4 :CUARTO
END

TO DETALLES.AQUI
  OUTPUT (LIST :DESCRIPCION :CONTENIDO
  :LISTA.SALIDAS :PELIGROS)
END

TO MOVER :DIR :LISTA
  IF PELIGRO? THEN PRINT [NO PUEDE
  AVANZAR HACIA ALLI] STOP
  IF EMPTY? :LISTA THEN PRINT [NO PUEDE
  AVANZAR HACIA ALLI] STOP
  MAKE "SALIDA FIRST :LISTA
  IF :DIR=FIRST :SALIDA THEN MOVER1 LAST
  :SALIDA STOP
  MOVER :DIR BUTFIRST :LISTA
END
```

Ahora MOVER impide todo movimiento hasta que PELIGROS sea [ ]. Al preparar los peligros de esta forma podemos utilizar el mismo peligro en numerosos cuartos, y desplazarlo de cuarto en cuarto simplemente alterando las descripciones de éstos.

Ahora podemos emplear los procedimientos que hemos desarrollado aquí para construir un juego de aventuras completo llamado *El santuario de Zoltoth*. En este juego el aventurero va en busca del cetro de Gilgesh, que ha sido robado por los sumos sacerdotes de Zoltoth y escondido en su templo subterráneo. El aventurero empieza el juego parado junto a la entrada de una cueva subterránea que conduce al santuario de Zoltoth. Cuando uno diseña un juego propio, puede comenzar por formular por escrito un escenario para un recorrido de éxito a través del juego, y estructurar el juego alrededor del mismo. Nosotros aquí no proporcionamos el escenario, de modo que usted está a tiempo de intentar construirlo si así lo desea.

El siguiente paso consiste en planificar el juego en términos de "cuartos", es decir, escenarios dentro del juego, su contenido y posiciones en relación unos con otros. Este trazado del mundo de fantasía se utiliza luego para definir los escenarios en el programa, dando las salidas posibles desde cada escenario. Los aventureros, a su vez, habrán de trazarse un mapa a medida que vayan avanzando.

Ahora necesitamos decidir acerca del vocabulario que se empleará en el juego: ¿qué palabras expresadas por el aventurero podrá comprender el programa? Nosotros aceptaremos:

1. Siete instrucciones compuestas por una sola palabra: EMPEZAR, MIRAR, N, S, E, O, e INVENTARIO (todas ellas fueron descritas en el capítulo anterior).
2. Instrucciones de dos palabras compuestas por un verbo seguido de un sustantivo. Los verbos son: COGER, DEJAR, EXAMINAR, MATAR, FROTAR y ABRIR. Los sustantivos son: ESPADA, COFRE, CETRO, ANILLO y SERPIENTE.

Todas las instrucciones se le digitan directamente al LOGO. Si se las reconoce, se obedecerán; pero si no se las reconoce, entonces el usuario obtendrá un mensaje de error del LOGO.

Sin embargo, sería mucho mejor ofrecer mensajes de error tales como "No conozco esa palabra", en vez de los mensajes de error estándares del LOGO. Para hacer esto necesitamos un bucle exterior que recoja las entradas, compruebe si son válidas y después las ejecute. He aquí una forma de



hacer esto para el vocabulario que hemos definido hasta ahora:

```
TO EMPEZAR
   MAKE "AQUI 1
   MAKE "INVENTARIO [ ]
   ESTABLECER.CUARTOS
   ASIGNAR.VARIABLES
   MIRAR
   JUEGO
END

TO JUEGO
   PRINT1 "INSTRUCCION:
   MAKE "ENTRADA REQUEST
   IF VALIDA? :ENTRADA RUN :ENTRADA ELSE
   PRINT [NO COMPRENDO]
   JUEGO
END

TO VALIDA? :INSTR
   IF ((COUNT :INSTR)=1)THEN OUTPUT
   VAL1? :INSTR
   IF ((COUNT :INSTR)=2)THEN OUTPUT
   VAL2? :INSTR
   OUTPUT "FALSE
END

TO VAL1? :INSTR
   IF MEMBER? FIRST :INSTR [INV O E S N
   MIRAR EMPEZAR] OUTPUT "TRUE
   OUTPUT "FALSE
END

TO VAL2? :INSTR
   IF ALLOF VALV? FIRST :INSTR VALN? LAST
   :INSTR OUTPUT "TRUE
   OUTPUT "FALSE
END

TO VALN? :SUSTANTIVO
   IF MEMBER? :SUSTANTIVO [ESPADA COFRE
   CETRO ANILLO SERPIENTE] OUTPUT "TRUE
   OUTPUT "FALSE
END

TO VALV? :VERBO
   IF MEMBER? :VERBO [COGER DEJAR EXAMINAR
   MATAR FROTAR ABRIR] OUTPUT "TRUE
   OUTPUT "FALSE
END
```

El programa

Primero debe entrar todos los procedimientos que ofrecimos en el capítulo anterior (véase p. 1255). Para empezar el juego, o para reiniciarlo en cualquier momento, digite EMPEZAR.

```
TO EMPEZAR
   MAKE "AQUI 1
   MAKE "INVENTARIO [ ]
   ESTABLECER.CUARTOS
   ASIGNAR.VARIABLES
   MIRAR
END
```

ESTABLECER.CUARTOS establece los cuartos de acuerdo al mapa.

```
TO ESTABLECER.CUARTOS
   MAKE "CUARTO.1 [[[UD ESTA PARADO JUNTO A
   LA ENTRADA] [DE UNA CUEVA]] [] [[E 2]] []]
   MAKE "CUARTO.2 [[[UD SE HALLA EN UNA
   CUEVA OSCURA Y HUMEDA]] [] [[S 3] [E 4]
   [O 1]] []]
   MAKE "CUARTO.3 [[[UD SE HALLA EN UNA
   CUEVA OSCURA Y HUMEDA]] [] [[N 2] [E 5]] []]
   MAKE "CUARTO.4 [[[UD ESTA EN UNA GRAN
   CAMARA SUBTERRANEA]] [] [[N 6] [S 5] [O 2]]
   [ATAQUE.SERPIENTE]]
   MAKE "CUARTO.5 [[[UD SE HALLA EN UNA
   CUEVA OSCURA Y HUMEDA]] [ESPADA] [[N 4]
   [O 3]] []]
   MAKE "CUARTO.6 [[[UD SE ENCUENTRA EN UNA
   HABITACION DE UN SANTUARIO SAGRADO] [EN
   UN NICHO DE LA PARED NORTE] [HAY UN
   ALTAR]] [] [[N 7] [S 4] [E 8]] [PUERTA]]
   MAKE "CUARTO.7 [[[UD ESTA PARADO JUNTO]
   [AL ALTAR DE ZOLTOTH EL DORADO] [ENCIMA
   DEL ALTAR ESTA ESCRITO:] ["NO DEJEIS
   ACERCAR NINGUN METAL BASE"]] [ANILLO] [[S
   6]] []]
   MAKE "CUARTO.8 [[[UD SE HALLA EN UNA
   CUEVA OSCURA Y HUMEDA]] [] [[S 10] [E 9] [O
   6]] [ATAQUE.SERPIENTE]]
   MAKE "CUARTO.9 [[[UD SE HALLA EN UNA
   CUEVA OSCURA Y HUMEDA]] [CETRO] [[S 11] [O
   8]] []]
   MAKE "CUARTO.10 [[[UD SE HALLA EN UNA
   CUEVA OSCURA Y HUMEDA]] [] [[N 8] [E 11]] []]
   MAKE "CUARTO.11 [[[UD SE HALLA EN LA
   SACRISTIA DEL] [SACERDOTE DE ZOLTOTH EL
   DORADO]] [CETRO] [[N 9] [O 10]] []]
END
```

Complementos al LOGO

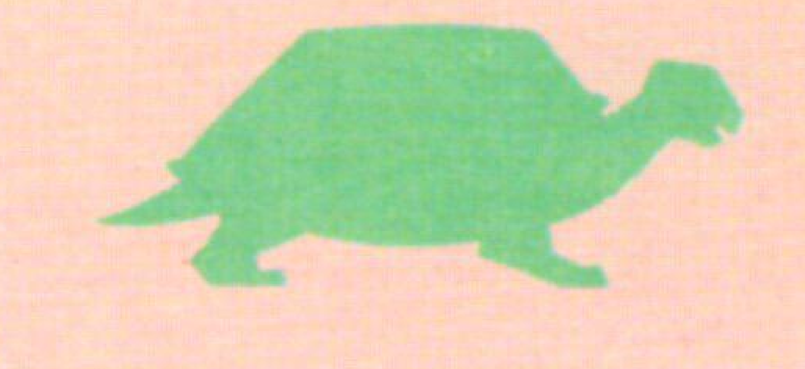
Algunas versiones de LOGO MIT no poseen EMPTY?, ITEM, COUNT ni MEMBER? En los capítulos anteriores (pp. 1234 y 1255) hemos ofrecido definiciones para las mismas. En las versiones LCSI utilice:

```
EMPTYP por EMPTY?
LISTP por LIST?
MEMBERP por MEMBER?
TYPE por PRINT1
AND por ALLOF
OR por ANYOF
```

La sintaxis IF en el LOGO LCSI queda demostrada mediante ese ejemplo:

```
IF EMPTYP :CONTENIDO [PRINT [NADA
   ESPECIAL]] [PRINT :CONTENIDO]
```

Si la condición es verdadera se ejecuta la primera lista después de la condición; si la condición es falsa, se ejecuta la segunda.
En el LOGO Atari utilice SE en lugar de SENTENCE, RL para REQUEST, y observe que ITEM no está implementada.





Sucesor del Vic-20

Demos una atenta mirada al Commodore 16, ordenador perteneciente a la "nueva generación" lanzada por su firma fabricante

Pensado posiblemente para sustituir al Vic-20, con su minúscula memoria de 3 583 bytes, el 16 comparte con el Plus/4 un BASIC muy potente, que complementa las instrucciones BASIC para tratamiento de discos existentes en máquinas como el CBM 8296 con un conjunto de instrucciones para gráficos y otras simples para sonido, haciendo al mismo tiempo un gesto simbólico hacia la programación estructurada con construcciones como DO...WHILE y LOOP...UNTIL...EXIT.

A diferencia del Plus/4, que posee un teclado y una carcasa radicalmente distintas a cualquier otra que haya producido Commodore con anterioridad, el 16 se suministra en una carcasa similar a la de los primeros modelos Vic y 64, si bien el esquema de color es diferente (color carbón, con teclas gris claro) y se ha modificado la disposición de las teclas. Las teclas son grandes, están bien situadas y poseen un tacto firme y profesional, como el de las máquinas que precedieron al 16. Una interesante innovación es la inclusión de una tecla HELP (en realidad la tecla de función F8), que ayuda al usuario, cuando el programa se interrumpe, con un informe de mensaje de error, visualizando la línea que contiene el error e iluminando de forma intermitente la parte incorrecta. En las líneas de sentencias múltiples, los caracteres se iluminan intermitentemente desde el error hasta el final de la línea; sería mucho más útil que la iluminación se restringiera al error propiamente dicho (p. ej., PRONT por PRINT).

Puntos en contra

El aspecto más controvertido tanto del 16 como del Plus/4 es el hecho de que por primera vez Commodore ha producido hardware que no es "compatible hacia arriba" con el producido previamente: ningún software del Vic ni del Commodore 64 funcionará en ninguna de las nuevas máquinas. Los conectores para palanca de mando y cassette son, asimismo, diferentes, con el primero apartándose del tipo de conector D de nueve patillas que por lo general se considera estándar. Los conectores para interface de disco y para pantalla son, no obstante, idénticos a los del 64.

De los cambios negativos y en cuanto atañe al programador de juegos, el más grave es que no existen sprites. Aunque el Vic-20 tampoco los posee, su utilización en el Commodore 64 (y en máquinas de la competencia) ha acostumbrado tanto a los usuarios a su empleo para la manipulación sencilla de formas gráficas, que ésta parece una curiosa omisión.

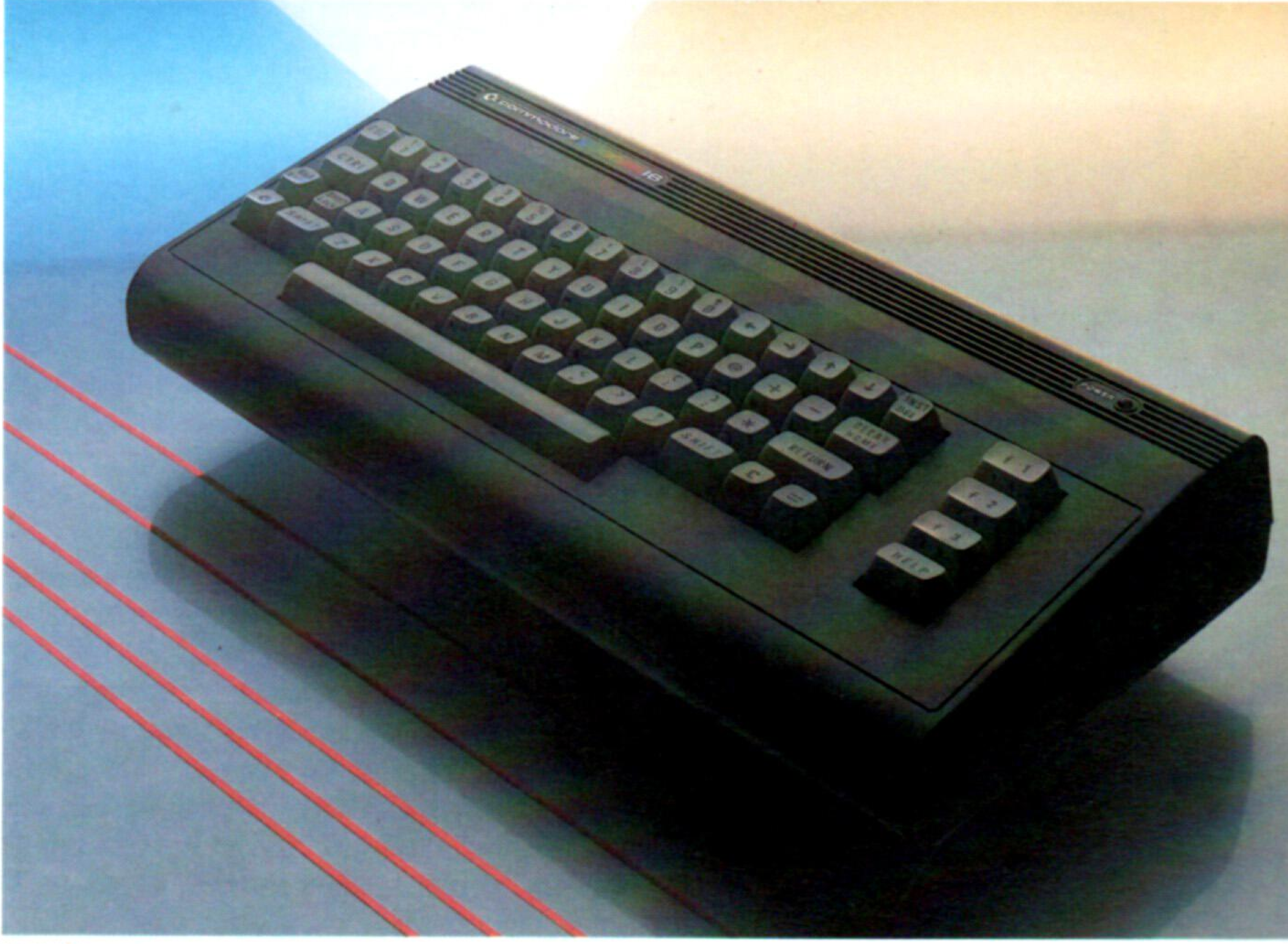
Al conectarse el equipo, la pantalla ofrece la ya familiar visualización Commodore, con la diferencia de que el BASIC indicado es la versión 3.5, y que hay 12 277 bytes de memoria disponibles. El BASIC 3.5 es en realidad la quinta versión que se escribe para máquinas Commodore. La versión original, 1.0, no contenía instrucciones para tratamiento de discos, y éstas eran muy torpes en la versión 2, que por algún motivo fue la versión que se incorporó en el Vic-20 y el Commodore 64. Hacia marzo de 1980, la versión 2 había sido superada por la versión 4, que es un BASIC muy eficaz escrito para los ordenadores de la serie Pet de 80 columnas, el 8032, 8096 y ahora el 8296.

La versión 3.5 es casi idéntica al BASIC 4, con unas cuantas instrucciones extras. En conjunto, la nueva versión posee unas 50 instrucciones y funciones más que las que ofrecía el Vic, incluyendo instrucciones de ayuda utilizadas para escribir y depurar programas, características de programación estructurada, instrucciones para gráficos y sonido.

La instrucción directa MONITOR invoca a Tedmon, el monitor residente (al cual también se puede acceder mediante SYS 4, como en la serie Pet). Éste emplea instrucciones mnemónicas de una sola letra: por ejemplo C, que compara dos secciones de la memoria e informa sobre las diferencias, y S, que guarda en cinta o en disco.

Las rutinas *kernal* básicamente no han sido modificadas. Estas rutinas, que se llaman desde código máquina, gobiernan el tratamiento de las rutinas de entrada y salida. No obstante, se ha agregado en $FF81 la función IOINIT del 64, para inicializar los dispositivos de entrada/salida (y especialmente los cartuchos de programas).

Commodore 16
Destinada a sustituir al Vic-20 en la línea de Commodore, la nueva máquina posee 16 K de memoria y un BASIC perfeccionado. El Commodore 16 es compatible con el Plus/4 tanto en software como en enchufes, pero no así con las máquinas Commodore más antiguas, el Vic-20 y el Commodore 64



Chris Stevens

Gráficos

La pantalla de alta resolución, trazada por mapa de bits, tiene unas dimensiones de 320 por 160 pixels, y la pantalla de colores múltiples da una resolución de 160 por 160. La instrucción de modo GRAPHIC es obviamente más fácil de invocar que las PEEK y POKE del 64, así como la pantalla dividida, si bien con ésta el texto queda limitado a las cinco líneas inferiores. No obstante, en una pantalla para gráficos se puede colocar texto en cualquier sitio mediante el empleo de la sentencia CHAR, de modo que

```
CHAR 1,0,0, "ESTA ES LA LINEA SUPERIOR"
```

se imprimirá a lo largo de la parte superior de la pantalla, siendo 1 el color seleccionado, y aludiendo los dos ceros a las posiciones de columna y fila. La serie se puede imprimir en video invertido si se la señala con un ',1'; esto se desactiva mediante ',0'. Cualquier error de sintaxis en alguna de las modalidades para gráficos devolverá al usuario a GRAPHIC 0, la pantalla de "texto puro".

La instrucción DRAW es una especie de compromiso entre las líneas rectas, bastante limitadas, disponibles en el Amstrad, y la DRAW MSX similar al LOGO. Por ejemplo, se puede dibujar un cuadrado mediante:

```
DRAW,10,10 TO 10,60 TO 60,60 TO 60,10 TO 10,10
```

En este caso no está definido el primer parámetro, de modo que el color del cuadrado será el último color establecido. Este color se puede modificar insertando el valor que interese. Existe asimismo una instrucción BOX (caja), que se utiliza específicamente para dibujar rectángulos especificando las posiciones de los cuatro vértices, con un parámetro de "relleno" para pintar la caja con un color.

La instrucción CIRCLE dibuja elipses, octágonos e incluso rombos y triángulos además de círculos propiamente dichos, a tenor de los parámetros que se especifiquen. Las formas no circulares se eligen especificando ángulos de 120° entre segmentos para un triángulo, 90° para un rombo y 45° para un octágono. El valor por defecto es 2°. PAINT rellenará la forma así creada, ya sea con el mismo color que la forma esbozada o bien con un color de primer plano definible, y las formas se pueden guardar (SAVE) o recuperar desde disco mediante el empleo de las instrucciones SSHAPE y GSHAPE.

Los colores se especifican desde BASIC asignándoles uno de entre 16 valores al fondo, primer plano, multicolor 1, multicolor 2 o borde, con un parámetro de brillo opcional de 0 a 7. El brillo por defecto es 7 (el más intenso). En todas las instrucciones para dibujar, el parámetro de color se debe escoger entre una de las cinco áreas ya definidas.

Sonido

Después del nivel de sofisticación de las instrucciones para sonido del chip SID (Sound Interface Device) del 64, el sonido de los canales del Commodore 16 es más bien una decepción, especialmente dado que la actual generación de máquinas competitivas, que utilizan el chip de sonido de General Instruments (Amstrad, MSX, Einstein), ofrece tres canales más ruido.

Sin embargo, la instrucción SOUND no exige la colocación (POKE) de cifras en las posiciones 54272



Bien venido el cambio

Al cabo de numerosas quejas formuladas por programadores en relación a las deficiencias de sus ROM de BASIC, finalmente Commodore ha editado una versión comparable a aquellas que ya son habituales en otros micros personales. Todas las adiciones son bien venidas. Inexplicablemente, los procedimientos definidos por el usuario se han omitido en esta versión de BASIC, que, con esta salvedad, ya es aceptable. Las nuevas instrucciones, incluidas las del BASIC 4, son las siguientes:

Funciones	Ayuda	Estructura	Gráficos	Del BASIC 4
RGR	AUTO	DO	GRAPHIC	DIRECTORY
RGCL	TRON	WHILE	SCNCLR	DSAVE
RLUM	TROFF	LOOP	PAINT	DLOAD
JOY	HELP	UNTIL	CHAR	HEADER
RDOT	MONITOR	EXT	BOX	SCRATCH
INSTR	DELETE	ELSE	CIRCLE	COLLECT
DEC	RENUMBER	PUDEF	GSHAPE	COPY
HEX$	KEY	USING	SSHAPE	RENAME
SOUND	ERR$		DRAW	BACKUP
VOL	TRAP		LOCATE	
	RESUME		COLOR	



COMMODORE 16

DIMENSIONES
76,2 × 203,2× 406,4 mm

CPU
MOS 7501, 0,89 a 1,76 MHz

MEMORIA
16 K de RAM (12 K de memoria para usuario), 32 K de ROM

PANTALLA
Texto: 25 filas de 40 columnas. Gráficos: 320 por 160 pixels. Cinco modalidades: texto, alta resolución, alta resolución con cinco líneas de texto, multicolor, multicolor con cinco líneas de texto, 15 colores × 8 niveles de brillo, más negro = 121 tonal.

INTERFACES
Puerta en serie Commodore, ranura para ampliación de cartucho de ROM/memoria, puerta para interface de unidad de cassette (8 patillas), 2 puertas para palanca de mando (8 patillas), salida para pantalla: compuesto/crominancia/brillo/audio, salida RF con interruptor para regulación alto/bajo, entrada alimentación eléctrica (9 V)

LENGUAJES DISPONIBLES
Intérprete de BASIC 3.5 en ROM, 75 instrucciones, incluyendo trazado completo de gráficos

TECLADO
Tipo máquina de escribir, 66 teclas, incluyendo 7 teclas de función reprogramables y HELP

VENTAJAS
BASIC avanzado, instrucciones excelentes para tratamiento de discos, instrucciones simples para sonido y para gráficos, fácil acceso a la pantalla para programación en lenguaje máq.

DESVENTAJAS
No es compatible con los equipos previos de CBM, razón por la cual dispone de muy poco software. Conectores de E/S incompatibles; no posee sprites

CPU 7501
Se trata de una variación Commodore del 6502

Chip TED
Este chip alberga el monitor residente e interactúa con la CPU para el control general del sistema

Reloj

Chip de BASIC
Aquí reside el intérprete de BASIC 3.5 de Commodore

Chris Stevens

a 54296, al igual que en el 64. Si se conoce la frecuencia de una nota, puede buscarse en una tabla del manual y utilizar la cifra proporcionada para definir la nota a reproducir. Por ejemplo:

```
SOUND 1,770,60
```

tocará la nota *la* (a una frecuencia de 440 Hz) durante 60 sesentavos de segundo (es decir, un segundo) en el canal uno.

El sonido más bajo que se puede ejecutar es *la* dos octavas por debajo de *do central* (110 Hz), y el más alto es *sol* dos octavas arriba de *do central* (1 575 Hz), dando una envergadura musical total de cuatro octavas. Hay disponibles dos canales de música (1 y 2) o un canal de música (1 o 2) y un canal de ruido blanco (3). Ambos canales están combinados, dado que la señal de salida de audio es mono, y no existe ninguna manera de separarlos.

El Commodore 16 es una máquina atractiva, con un BASIC muy avanzado y buenas instrucciones para gráficos, pero sus facilidades para sonido son bastante primitivas, aun comparándolas con las del Vic-20, si bien en la nueva máquina son más fáciles de ejecutar.

En el momento de su lanzamiento, era muy poco el software existente para el Commodore 16. A los poseedores de Vic 20 que deseen superar y poner al día sus expectativas en informática adquiriendo esta máquina, se les debe advertir que en la misma no se ejecutarán sus antiguos programas.

La línea de la trama

Llegados a este punto, desarrollaremos una utilidad que nos permitirá formatear la salida a la pantalla de los juegos creados

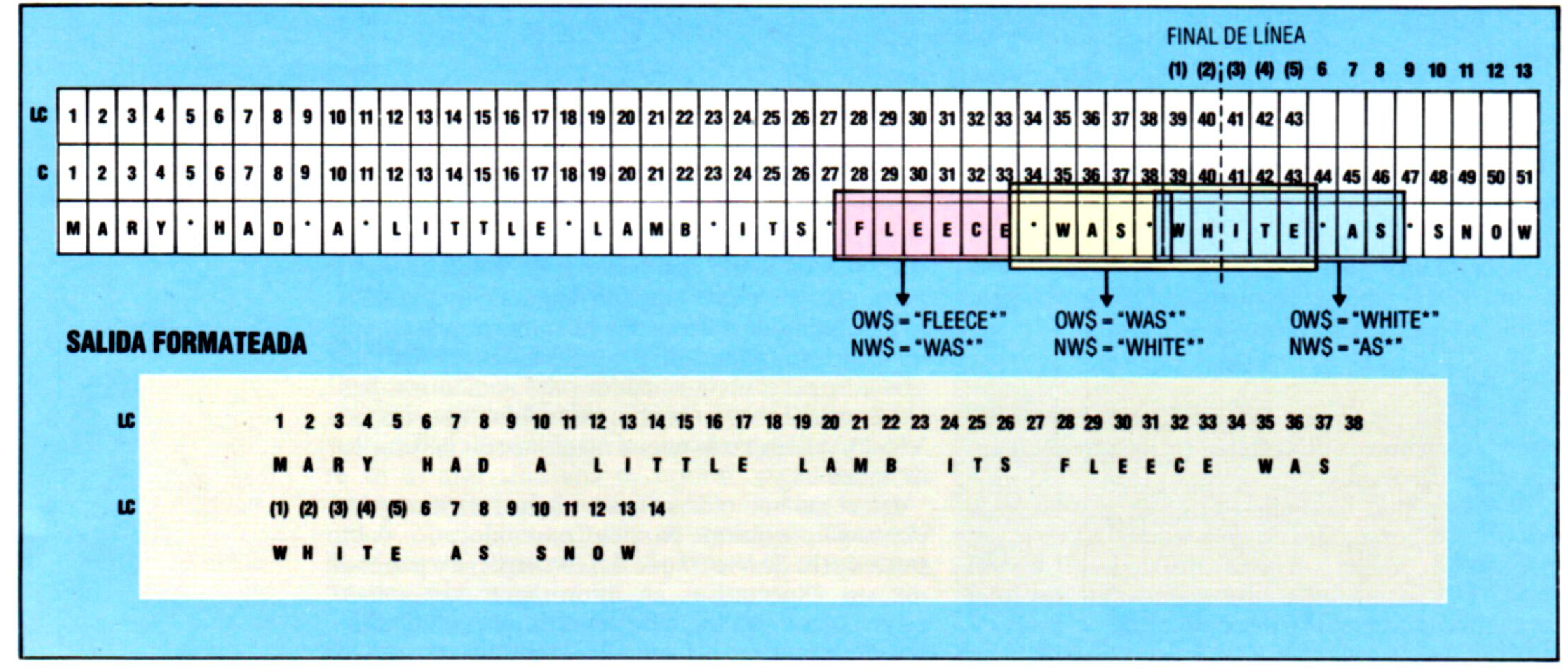
Dado que tanto *Digitaya* como *El bosque encantado* son aventuras basadas en texto, emplean palabras para describir los escenarios y los acontecimientos. Pasar esta información a la pantalla utilizando sentencias PRINT sería poco elegante. Por ejemplo, una sentencia PRINT cuya longitud supere a la de una línea de la pantalla continúa en la línea siguiente, con frecuencia dividiendo en dos aquellas palabras que quedan al final de la línea de pantalla. Una laboriosa forma de abordar este problema sería considerar cada sentencia PRINT del programa de forma individual y formatear "manualmente" la salida de modo que no se cortaran las palabras del final de cada línea. Ello no representaría mayor problema si sólo hubiera de hacerse en unas pocas ocasiones, pero en un programa para un juego de aventuras ello será necesario en innumerables ocasiones. La alternativa es diseñar una rutina que nos formatee la salida. Para emplear una rutina de este tipo debemos ser capaces de pasar la frase que deseamos formatear a la rutina a través de una variable en serie, y la rutina deberá encargarse del formateado y de la salida.

Tanto *Digitaya* como *El bosque encantado* utilizan una rutina especial para formatear su salida, de modo que antes de seguir describiendo la programación del juego será mejor que analicemos cómo trabaja esta rutina. Éste es el listado para *El bosque*...

```
5500 REM **** S/R FORMATEADO DE SALIDA ****
5510 LC=0:        REM CONTADOR CAR/LINEA
5520 OC=1:        REM VALOR INICIAL CONTADOR ANTIGUO
5530 OWS="":      REM VALOR INICIAL PALABRA ANTIGUA
5540 LL=40:       REM LONGITUD LINEA
5550 SN$=SN$+ " FICTICIA "
5560 PRINT
5570 FOR C=1 TO LEN(SN$)
5580 LC=LC+1
5590 IF MID$(SN$,C,1)=" " THEN GOSUB5800
5600 NEXT C
5605 PRINT
5610 RETURN
5620 :
5800 REM ** S/R CONTROL FINAL DE LINEA **
5810 NW$ = MID$(SN$,OC, C-OC+1):REM NUEVA PALABRA
5820 IF LC<LL THEN PRINTOW$;:GOTO5840
5830 PRINTOW$:LC=LEN(NW$)
5840 OC=C+1:OW$=NW$
5850 RETURN
```

En primer lugar, la rutina busca a través de la frase, que le ha sido pasada por la variable SN$, un carácter espacio. Cada vez que encuentra un espacio se llama a la subrutina de la línea 6020. Esta subrutina lleva a cabo varias tareas importantes. Utilizando OC para indicar el comienzo de una palabra (inicialmente, OC está establecida en 1), y C para llevar el registro del carácter que está siendo objeto del examen, se puede aislar la palabra que se ha encontrado antes del espacio utilizando MID$ y almacenarla en NW$ (por *New Word*: palabra nueva). Antes de visualizar en la pantalla el contenido de NW$, el mismo se transferirá a OW$.

Se emplea un contador de línea, LC, para contar cuántos caracteres se han utilizado hasta el momento en cualquier línea dada, cantidad que se verifica en la línea 6040 para asegurar que sea inferior a la longitud de línea permitida, LL. De ser éste el caso, entonces se imprime OW$, seguida de un punto y coma para asegurar que cualquier salida que venga a continuación continuará en la misma línea. Si LC sí fuera mayor que LL, nuevamente se imprimiría OW$, pero en esta ocasión omitiendo el punto y coma (y, por tanto, cualquier salida que venga a continuación empezará en una línea nueva). Además, LC, el contador de línea, se vuelve a establecer en la longitud de la palabra nueva.



Visualización en formación
La rutina para formatear la pantalla que utilizan *Digitaya* y *El bosque encantado* permite formatear cualquier salida en pantalla de modo que no se produzcan cortes de palabras. Mediante el empleo de las variables OW$ y NW$, la rutina va "buscando" con una palabra de antelación respecto a la que se está visualizando. Si la siguiente palabra va a superar la longitud de línea establecida, se omite el punto y coma que suprime un retorno de carro, haciendo que se inicie una línea nueva

Veamos ahora cómo funciona esta subrutina en la práctica. La rutina explora la frase a formatear, en busca de un espacio. Cuando encuentra uno, se vuelve a designar los caracteres entre el espacio y la última palabra hallada como si formaran una nueva palabra. La rutina, en efecto, busca palabras con una palabra de adelanto respecto a la que se está visualizando. La rutina comprueba si la longitud máxima de palabra se ha superado cuando agrega a la línea de pantalla el nuevo término. Si así fuera, la rutina hace que se inicie una línea nueva. Por consiguiente, se evita la separación de las palabras del final de cada línea. La adición de "FICTICIA" al final de la frase es importante, ya que representa una última palabra a almacenar en NW$. Los espacios antes y después de " FICTICIA " son significativos: el primero la convierte en una palabra separada y el último proporciona un espacio final a detectar por la rutina.

Tomemos a modo de ejemplo la oración *"Mary had a little lamb; its fleece was white as snow"* (Mary tenía un corderito; su vellón era blanco como la nieve). La anchura de pantalla que utilizaremos es de 40 caracteres. Si no se formateara la oración, la palabra *white* se separaría en dos, con las letras *ite* empezando una línea nueva. La rutina de formateado, sin embargo, toma dos palabras de la oración cada vez. Si consideramos las dos que preceden a *white*, entonces *fleece* se almacenará en OW$ y *was* en NW$. Habiendo comprobado que el contador, LC, no supera la cantidad de 40, se imprime OW$, seguida de un punto y coma; *was* se transfiere entonces de NW$ a OW$ y la rutina continúa explorando la frase, y se encuentra con el término *white*. En este punto, el contador LC excede de 40, lo que indica que *white* cae en un final de línea. Dada esta situación, OW$ (que ahora contiene la palabra *was*) sí se visualiza igualmente pero sin el punto y coma. Además, se reestablece el contador LC en la cantidad de caracteres de este término. La palabra *white* se transfiere a OW$, para la subsiguiente impresión de una nueva línea.

Comprobación de la rutina

Para comprobar la rutina, la emplearemos para formatear y visualizar la descripción inicial de la historia. Podemos ensamblar una frase de hasta 48 caracteres, utilizando la variable SN$, y llamar a la subrutina de formateado. Digite estas líneas:

```
1000 REM **** S/R HISTORIA HASTA AHORA ****
1010 SN$="BIENVENIDO AL BOSQUE ENCANTADO"
1020 GOSUB5500:REM FORMATEAR
1030 PRINT
1040 SN$="AL DESPERTARTE DE UN PROFUNDO SUEÑO, EL "
1050 SN$=SN$+"SUELO DEL BOSQUE ESTA SUAVE Y SECO. "
1060 SN$="TU NO SABES COMO HAS LLEGADO HASTA AQUI "
1070 SN$=SN$+"PERO SABES QUE DEBES LLEGAR AL "
1080 SN$=SN$+"POBLADO QUE HAY A LAS AFUERAS DEL BOSQUE
     PARA "
1090 SN$=SN$+"ESTAR A SALVO."
1100 GOSUB5500:REM FORMATEAR
1110 PRINT
1120 SN$="ECHAS UNA MIRADA A TU ALREDEDOR PARA ORIENTARTE "
1130 GOSUB5500:REM FORMATEAR
1140 PRINT:PRINT"PULSA CUALQUIER TECLA PARA COMENZAR"
1150 GET A$:IF A$="" THEN 1150
1160 PRINTCHR$(147):REM LIMPIAR PANTALLA
1170 RETURN
```

Para llamar a la subrutina "Historia hasta ahora" hemos de emplear estas líneas:

```
205 GOSUB 1000: REM HISTORIA HASTA AHORA
990 END
```

Listados para "Digitaya"

```
1110 GOSUB1250:REM HISTORIA HASTA AHORA
1270 END

1290 REM **** HISTORIA HASTA AHORA ****
1300 SN$="BIEN VENIDO A 'DIGITAYA'
1310 GOSUB5880:REM FORMATEAR
1320 PRINT
1330 SN$="MIENTRAS LA MAQUINA SUSURRA QUEDAMENTE,
     ECHAS UNA MIRADA EN DERREDOR."
1340 SN$=SN$+" HACIA EL NORTE Y HACIA EL SUR SE EXTIENDE
     UNA ANCHA AUTOPISTA."
1350 SN$=SN$+" TU MISION CONSISTE EN ENCONTRAR AL
     MISTERIOSO DIGITAYA"
1360 SN$=SN$+" Y SACARLO A TRAVES DE UNA DE LAS
     PUERTAS DE SALIDA PARA PONERLO A SALVO."
1370 SN$=SN$+".. PERO POR CUAL DE LAS PUERTAS ?"
1380 GOSUB5880
1390 PRINT:PRINT"PULSA UNA TECLA PARA COMENZAR"
1400 GETA$:IFA$=""THEN1400
1410 PRINTCHR$(147):REM LIMPIAR PANTALLA
1420 RETURN

5880 REM **** S/R IMPRESION FORMATEADA ****
5890 LC=0:          REM CONTADOR CAR/LINEA
5900 OC=1:          REM CONTADOR ANTIGUO
5910 OW$="":          REM PALABRA ANTIGUA
5920 LL=40:REM LONGITUD LINEA ANTIGUA
5930 SN$=SN$+" FICTICIA "
5940 PRINT
5950 FOR C+1 TO LEN(SN$)
5960 LC=LC+1
5970 IF MID$(SN$,C,1)=" " THENGOSUB6020
5980 NEXTC
5990 PRINT
6000 RETURN
6010 :
6020 REM **** S/R COMPROBACION FINAL DE LINEA ****
6030 NW$=MID$(SN$,OC,C-OC+1)
6040 IF LC<LL THENPRINTOW$;:GOTO6060
6050 PRINTOW$:LC=LEN(NW$)
6060 OC=C+1:OW$=NW$
6070 RETURN
```

Complementos al BASIC

Spectrum:

Para el listado de *Digitaya*, introduzca estas modificaciones en la Rutina de Formateo:

Reemplace SN$ por S$, OW$ por O$, NW$ por N$

```
5920 LET LL=32:REM LONGITUD LINEA
PANTALLA
5970 IF S$(C TO C)="" THEN GOSUB 6020
6030 LET N$=S$(OC TO C)
```

En la subrutina Historia... sustituya SN$ por S$

```
1400 IF INKEY$="" THEN 1400
1410 CLS
```

Para el listado de *El bosque encantado*, reemplace los nombres de las mismas variables en serie y modifique estas líneas:

```
5540 LET LL=32:REM LONGITUD LINEA
     PANTALLA
5590 IF S$(C TO C)="" THEN GOSUB 5800
5810 LET N$=S$(OC TO C)
```

y

```
1150 IF INKEY$="" THEN 1150
1160 CLS
```

BBC Micro:

Para la subrutina Historia Hasta Ahora, se deben introducir en *Digitaya* estos cambios:

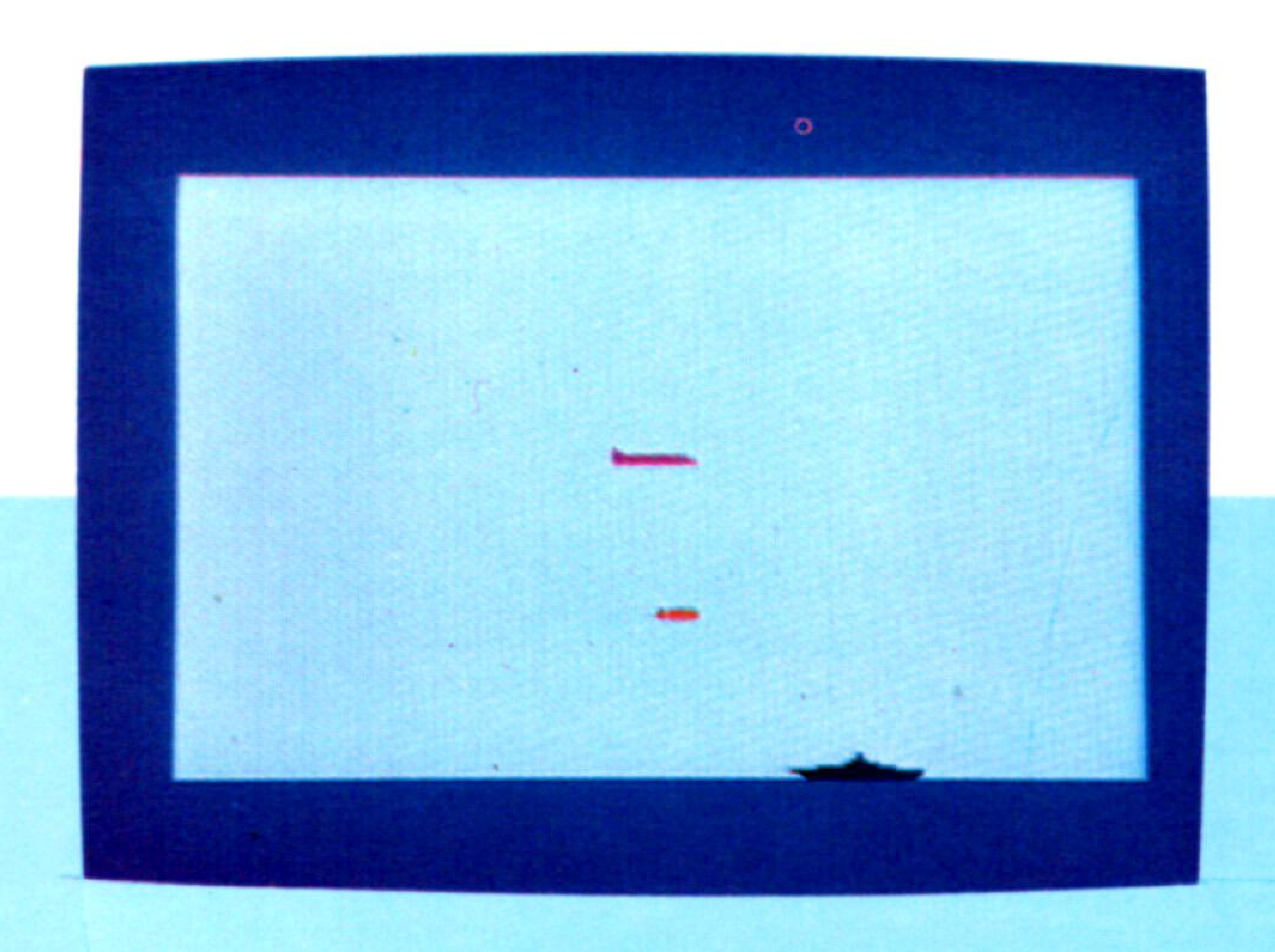
```
1095 MODE 1
1400 A$=GET$
1410 CLS
```

y en *El bosque encantado*:

```
1160 CLS
```

Exocet

El mortífero misil Exocet, que alcanzó notoriedad durante la guerra de las Malvinas, es el protagonista de este juego escrito para el Commodore Vic 20



Un portaaviones enemigo se ha aventurado por las aguas territoriales de su país y hace caso omiso a sus requerimientos. A los mandos de su Mirage 2000, debe destruirlo antes de que constituya una amenaza para su base. Para disparar pulse una tecla cualquiera.

```
10 REM **********
20 REM * EXOCET *
30 REM **********
35 GOSUB 2000
40 GOSUB 1000
100 PRINT TAB(A)A$
105 IF B>17 THEN PRINT TAB(18)N2$;CHR$(19);:BB
    =0:GOTO 120
110 PRINT TAB(B)B$;
120 A=A-1
130 IF A<0 THEN PRINT D$+N2$+CHR$(19);:A=19
140 BB=BB+0.2
150 B=INT(BB)
160 GET X$
170 IF X$<>"" AND EX=0 THEN EX=A+7923:NX=NX-1
180 IF EX<>0 THEN 300
190 FOR I=1 TO 10
200 NEXT I
210 GOTO 100
300 EX=EX+21
310 IF EX>8184 THEN 400
320 C=PEEK(EX)
330 IF C<>32 THEN GOSUB 700
340 POKE EX-21,CN
350 POKE EX,CX
360 POKE EX+M,XC
370 GOTO 100
400 POKE EX-21,CN
410 EX=0
420 IF NX=0 THEN 500
430 GOTO 100
500 PRINT CHR$(147)
505 IF S>R THEN R=S
510 GET X$
515 IF X$<>"" THEN 510
520 PRINT:PRINT:PRINT
525 PRINT CHR$(31)
530 POKE 36869,240
535 PRINT TAB(5)"PUNTOS :"S
540 PRINT:PRINT:PRINT
550 PRINT TAB(5)"RECORD :"R
560 PRINT:PRINT:PRINT
570 PRINT TAB(5)"OTRA ?"
580 FOR I=1 TO 300:GET X$:NEXT I
600 GET X$
610 IF X$="" THEN 600
620 IF X$<>"N" THEN :POKE 36869,254:GOTO40
630 PRINT CHR$(147);
640 POKE 36879,27
650 END
700 POKE EX-21,CN
705 S=S+10
710 POKE EX,6
715 POKE EX+M,2
720 FOR I=1 TO 30
725 X=INT(RND(TI)*4)
730 Y=INT(RND(TI)*6)
740 XY=7680+(22-Y)*22+B+X
750 POKE XY,6
760 POKE XY+M,2
770 NEXT
780 FOR I=1 TO 200:NEXT
800 NX=NX+1:PRINT CHR$(147);:GOTO 100
1000 PRINT CHR$(147);
1010 M=30720
1020 B$=CHR$(144)+CHR$(32)+CHR$(64)+CHR$(65)+CHR$(66)
     +CHR$(19)
1030 A=19:S=0:BB=0:B=0:D$=""
1060 FOR I=1 TO 10
1070 D$=D$+CHR$(17)
1080 NEXT
1085 D1$=D$+CHR$(17)
1090 A$=CHR$(156)+D$+CHR$(67)+CHR$(68)+CHR$(32)+
     CHR$(19)+D1$+D$
1100 N2$=CHR$(32)+CHR$(32)+CHR$(32)
1120 EX=0:CX=5:XC=2:NX=20:CN=32:RETURN
2000 PRINT CHR$(147);
2010 POKE 36869,254
2020 POKE 52,24:POKE 56,24
2040 FOR I=0 TO 55
2050 READ A
2060 POKE 6144+I,A
2070 NEXT
2080 FOR I=0 TO 7
2090 POKE 6400+I,0
2100 NEXT
2110 POKE 36879,30:RETURN
3000 DATA 0,0,0,0,7,255,255,127
3010 DATA 16,16,56,252,255,255,255,255
3020 DATA 0,0,0,0,224,255,252,248
3030 DATA 0,0,0,0,0,63,127,255
3040 DATA 0,0,0,1,3,255,255,255
3050 DATA 0,0,0,0,125,255,125,0
3060 8,33,128,10,0,40,0,16
```



Rupturas

Seguimos con nuestra tarea de diseño y escritura de un programa depurador de errores

Nos quedan todavía por definir dos subrutinas del módulo Puntos De Ruptura, una para eliminar puntos de ruptura previamente insertados y otra para restaurar el opcode original donde hayamos colocado un opcode SWI temporal. Debemos analizar en primer lugar la rutina que llamaremos Desinsertar Puntos de Ruptura (de una tabla de dichos puntos).

Se ha posibilitado la colocación de hasta 16 puntos en Tabla-Puntos-Ruptura (BPTAB). Para eliminar uno de ellos debemos obtener su número, que servirá de desplazamiento (dentro del intervalo del 0 al 15) en la tabla. La entrada de la tabla se elimina desplazando todas las entradas subsiguientes un lugar hacia atrás en la tabla (dos bytes) y decrementando Número-Punto-Ruptura.

Desinsertar-Puntos-De-Ruptura

Data:
- **Número-Puntos-Ruptura** es un valor de 8 bits
- **Número-Punto-Ruptura** es un contador de 8 bits
- **Tabla-Puntos-Ruptura** es una tabla de direcciones de 16 bits
- **Entrada-a-Descartar** es un desplazamiento de 8 bits (con valores entre 1 y 16)

Proceso: Desinsertar-Puntos-De-Ruptura

Decrementar Número-Puntos-Ruptura
If (si) Entrada-a-Descartar<=Números-Puntos-Ruptura (penúltimo) THEN
 For (desde) Número-Punto-Ruptura=Entrada-a-Descartar *To* (hasta) Número-Puntos-Ruptura (penúltimo)
 Llevar Tabla-Puntos-Ruptura (Número-Punto-Ruptura+1) a Tabla-Puntos-Ruptura (Número-Punto-Ruptura)
 Llevar Valores-Sustituidos (Número-Punto-Ruptura+1) a Valores-Sustituidos (Número-Punto-Ruptura)
 Endfor
Endif

Fin de proceso

El parámetro Entrada-a-Descartar puede ser pasado en B. El contador Número-Punto-Ruptura puede también colocarse en B, y adquirirá automáticamente su correcto valor inicial. Después de compararlo con Número-Puntos-Ruptura, debe decrementarse para formar el desplazamiento en la tabla de Valores-Sustituidos de ocho bits y posteriormente se desplaza (se multiplica por dos) para formar un desplazamiento en la Tabla-Puntos-Ruptura de 16 bits. El desplazamiento de 8 bits se puede guardar en B y el de 16 bits en A. Las direcciones de las entradas en ambas tablas pueden estar en X e Y, de modo que podemos emplear un autoincremento para recorrer la tabla. La entrada de 16 bits puede ser desplazada a través de U, pero la de 8 bits tendrá que utilizar A de nuevo.

El último proceso empleado en este módulo elimina físicamente un punto de ruptura sustituyendo el opcode SWI con el código original en la Tabla de Valores-Sustituidos.

Eliminar-Punto-Ruptura

Data
- **Número-Punto-Ruptura** es un desplazamiento de 8 bits en la Tabla-Puntos-Ruptura

Proceso:

Tomar el valor que se halla en Valores-Sustituidos (Número-Punto-Ruptura)
Almacenarlo en la dirección indicada en Tabla-Puntos-Ruptura (Número-Punto-Ruptura)

Asumiremos que el parámetro Número-Punto-Ruptura es pasado en B de la forma habitual como número entre 1 y 16, que ha de convertirse para que funcione como un desplazamiento en las tablas.

Estamos ya en la etapa de construir un módulo para ejecutar las ocho órdenes de una letra simple que operan el sistema (véase p. 1238). Varias de estas órdenes pueden ejecutarse por medio de las rutinas que ya hemos escrito. No obstante, para ser completos y obtener una adecuada estructura modular incorporaremos llamadas a dichas rutinas desde este módulo.

La orden B, insertar un punto de ruptura, se consigue plenamente con la rutina Inserción-Puntos-Ruptura (BP01). En este módulo, pues, sólo necesitamos escribir:

```
CMDB BRA BP01
```

La orden U, desinsertar un punto de ruptura, casi se cumple con la rutina que acabamos de escribir (BP04). Pero primero debemos tomar la dirección del punto de ruptura que hay que descartar buscando en la Tabla-Puntos-Ruptura para encontrarla. Si no se encuentra allí, se ignora la orden; si se encuentra, podemos pasar el desplazamiento a la subrutina en BP02.

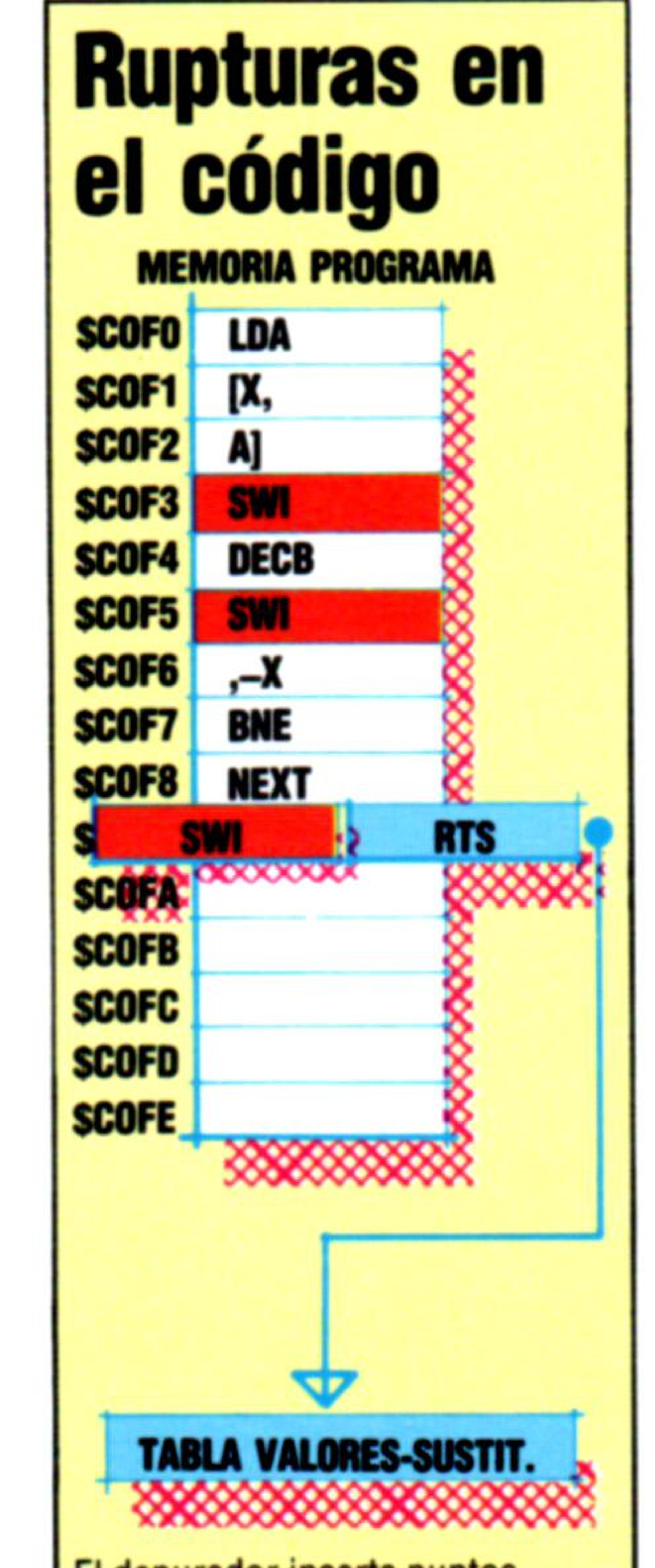
La orden U

Data:
- **Aviso** a visualizar
- **Dirección-Punto-Ruptura** es la entrada
- **Tabla-Puntos-Ruptura**
- **Número-Punto-Ruptura**

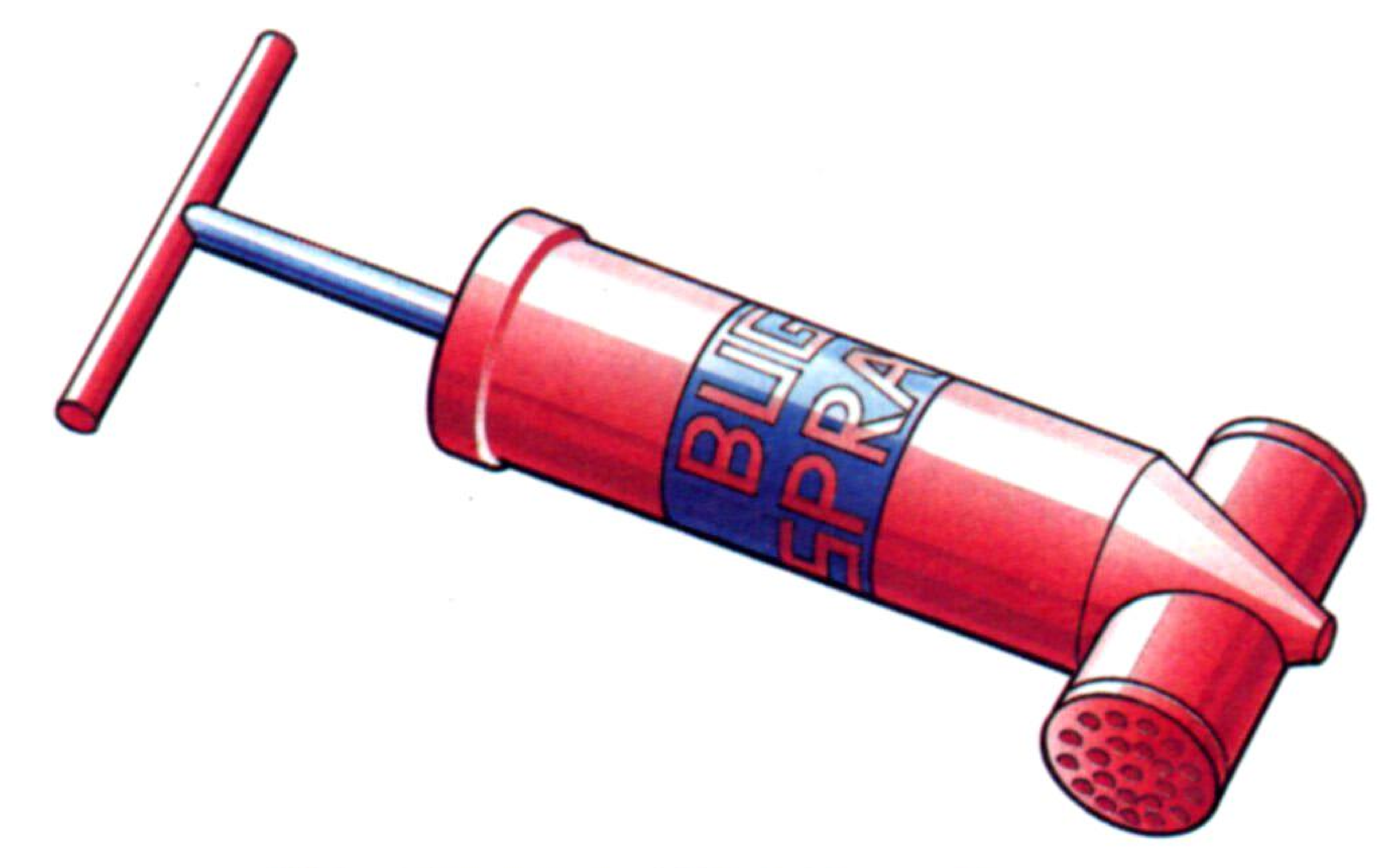
Proceso:

Visualizar el aviso
Tomar Dirección-Punto-Ruptura
Poner a 16 el Número-Punto-Ruptura

Rupturas en el código



El depurador inserta puntos de ruptura en el código objeto bajo comprobación guardando primero el código de esa dirección en la Tabla de Valores-Sustituidos e incrementando el contador de Número-Puntos-Ruptura, para escribir encima del contenido del byte del punto de ruptura por el opcode SWI. La Tabla de Valores-Sustituidos es un hacinamiento de valores que pueden tomarse de cualquier posición. Cuando se quita un punto de ruptura el Valor Sustituido correcto se vuelve a copiar de la tabla en la memoria que contiene el programa, y el byte sobrante se elimina moviendo hacia abajo todos los bytes de la tabla que están encima de él, decrementando, por último, el contador de Número-de-Puntos-Ruptura



```
While (siempre que) Tabla-Puntos-Ruptura
   (Número-Punto-Ruptura) <> Dirección-Punto-
   Ruptura
   y Número-Punto-Ruptura > 0
   Decrementar Número-Punto-Ruptura
Si es encontrada entonces
   Desinsertar-Punto-Ruptura
```

La Dirección-Punto-Ruptura se puede guardar en Y, dejando X disponible para su empleo como índice de la tabla. Número-Punto-Ruptura puede guardarse en B.

La orden D, visualizar los puntos de ruptura, se realiza con la rutina que etiquetamos DISPBP. Se accede a ella con una simple bifurcación a subrutina:

```
CMDD   BRA   DISPBP
```

La orden S, iniciar la ejecución del programa, es algo más complicada, puesto que es aquí donde se insertan los puntos de ruptura. El opcode de la instrucción SWI debe ser insertado en cada dirección dentro de la Tabla-Puntos-Ruptura, y el opcode que ya se encuentra allí se coloca en Valores-Sustituidos. Una vez hecho esto, el control se transferirá a la dirección de inicio del programa. Debemos observar también que el siguiente punto de ruptura es el número 1. He aquí el proceso completo para el inicio del programa:

Orden S

Data:

Número-Puntos-Ruptura es un valor de 8 bits
Tabla-Puntos-Ruptura
Valores-Sustituidos
Número-Punto-Ruptura es un contador de 8 bits
Siguiente-Punto-Ruptura es un valor de 8 bits
Opcode-SWI es un valor de 8 bits
Inicio-Dirección es la dirección de 16 bits inicial del programa que tratamos de depurar

Proceso:

```
Poner Número-Punto-Ruptura a Número-Puntos-
Ruptura
While (siempre que) Número-Punto-Ruptura > 0
   Establecer-Punto-Ruptura (Número-Punto-
   Ruptura)
   Decrementar Número-Punto-Ruptura
   Endwhile
Poner Siguiente-Punto-Ruptura a 1
Saltar a Inicio-Dirección
```

Para esto último empleamos la rutina Establecer-Punto-Ruptura, ya codificada, que requiere tener en A el Número-Punto-Ruptura (menos 1, para que pueda servir como desplazamiento en las tablas). Decrementaremos, por conveniencia, a A antes de llamar a la rutina Establecer-Punto-Ruptura. Damos finalmente la rutina codificada.

El modo de concluir esta rutina exige una pequeña aclaración. Cuando se está ejecutando el programa a depurar no se necesitan ítems adicionales en la pila, por lo que habremos de cerciorarnos de que la pila está vacía cuando se transfiere el control al programa. Podríamos borrar todo ítem superfluo de la pila en el nódulo principal, pero si es llamada esta rutina por medio de BSR (para mantener la consistencia con las otras órdenes) la dirección de retorno deberá colocarse en la pila. Y si la dejamos allí en una sesión larga (en la que el programa puede ser reiniciado repetidas veces) la pila puede crecer desmesuradamente. La solución empleada consiste en quitar la dirección de la pila en el momento en que el control es devuelto al programa. Y esto se hace sustituyendo la dirección de retorno colocada en la pila por la dirección inicial. Entonces la RTS hace saltar la dirección de retorno, ya convertida en dirección inicial, fuera de la pila, transfiriendo el control al mismo tiempo que restablece la pila.

En esta lección examinaremos finalmente la orden M, inspeccionar y cambiar posiciones de memoria. Se trata de obtener como entrada una dirección y de visualizar el contenido de esa dirección en la pantalla. El usuario puede entonces introducir un número hexa de dos dígitos para colocarlos en esa posición, o sencillamente un Return. En cualquier caso pasamos a la siguiente posición en la memoria. El usuario puede detener el proceso entrando un punto. La rutina GETHX2 se codificó teniendo esto presente, permitiendo la entrada de dos dígitos hexa o bien de un punto o de un Return.

La orden M

Data:

Posición-Actual es la dirección de 16 bits de la posición que queremos inspeccionar
Valor-Actual se encuentra en Posición-Actual y es de 8 bits
Valor-Nuevo para Posición-Actual; también de 8 bits

Proceso:

```
Tomar Posición-Actual
Repeat
   Visualizar Valor-Actual
   Tomar Valor-Nuevo
   If Valor-Nuevo no es un punto then
      If Valor-Nuevo no es un Return
         Almacenar Valor-Nuevo en Posición-Actual
      Endif
      Incrementar Posición-Actual
      Visualizar Posición-Actual
   Endif
Until (hasta que) Posición-Actual sea un punto
```

Para esta rutina se almacenará la Posición-Actual en X, empleándose el registro B tanto para Valor-Actual como para Valor-Nuevo. Por su parte, A sirve de flag que indica cuál de las tres posibilidades (un hexa, un punto o un Return) se ha introducido.

Nos quedan por diseñar y codificar las restantes tres órdenes, G, R y Q. Pero para ello se necesita emplear un mecanismo de interrupción que habremos de analizar detenidamente. De ello trataremos en la próxima lección junto con el diseño del módulo principal del programa depurador de errores.



Rutina Desinsertar

```
BP04    PSHS  A,B,X,Y,U       Salva regs. empleados
        DEC   NUMBP,PCR       Decrementa Número-Puntos-
                              Ruptura
IF02    CMPB  NUMBP,PCR       Si Entrada-a-Descartar <
        BGT   ENDF02          Número-Puntos-Ruptura
        DECB                  Convierte B en un despl.
        TFR   B,A             Copia B en A
        LSLA                  Convierte A en un despl.
        LDX   BPTAB,PCR       Dirección base de Tabla-Puntos-
                              Ruptura
        LEAX  A,X             Tabla-Puntos-Ruptura (Número-
                              Punto-Ruptura)
        LDY   REMTAB,PCR      Dirección base de Valores-
                              Sustituidos
        LEAY  B,Y             Valores-Sustituidos (Número-
                              Punto-Ruptura)
FOR00   LDU   2,X             Toma la entrada de Tabla-
                              Puntos-Ruptura a mover
        STU   ,X++            La mueve un lugar hacia atrás
        LDA   1,Y             Toma la entrada de Valores-
                              Sustituidos que hay que mover
        STA   ,Y+             La mueve un lugar hacia atrás
        INCB
        CMPB  NUMBP,PCR       ¿Ultima?
        BLT   FOR00           La siguiente
ENDF02  PULS  A,B,X,Y,U,PC    Restaurar y retornar
```

Rutina Eliminar-Punto-Ruptura

```
BP05    PSHS  A,B,X
        DECB                  Conversión en despl. para
                              Valores-Sustituidos
        LDX   REMTAB,PCR      Dirección base de Valores-
                              Sustituidos
        LDA   B,X             Toma el valor a mover
        LSLB                  Conversión en despl. para
                              Tabla-Puntos-Ruptura
        LDX   BPTAB PCR       Dir. base de Tabla-Puntos-
                              Ruptura
        STA   [B,X]           Almac. en tabla según dir.
        PULS  A,B,X,PC        Restaurar y retornar
```

La orden U

```
PROMPT  FCB   '>
CMDU    PSHS  A,B,X,Y         Guarda regs. empleados
        LDA   PROMPT,PCR
        BSR   OUTCH           Visualiza el aviso
        BSR   GETADD          Toma Dirección
        TFR   D,Y             Pone Dir.-Punto-Ruptura en Y
        LDB   MAXBP,PCR       Número máximo de Puntos-
                              Ruptura (16)
        LDX   BPTAB,PCR       Dir. base de Tabla-Puntos-Rupt.
        TFR   B,A
        LSLA                  A es un desplazamiento al final
                              de Tabla-Puntos-Ruptura
        LEAX  A,X             X señala ahora más allá del final
                              de la tabla
        TSTB                  Activa los flags según contenido
                              de B
WHIL02  BLE   ENDW02          While B>0
        CMPY  ,--X            (Recuerde que X se decrementa
                              primero)
        BEQ   ENDW02          y (and) Dir.-Puntos-Rupt. no se
                              encuentra en la tabla
        DECB                  Decrementa Número-Punto-
        BRA   WHIL02          Ruptura
ENDW02  TSTB                  Se encuentra si B>0
IF03    BLE   ENDF            Si se encuentra, entonces
        BSR   BP04            Desinsertar-Punto-Ruptura
ENDF03  PULS  A,B,X,Y
```

La orden S

```
START   RMB   2               Dirección-Inicio
CMDS    LDA   NUMBP,PCR       Pone Número-Punto-Ruptura a
                              Número-Puntos-Ruptura
WHIL03  TSTA                  Prueba el valor de Número-
                              Punto-Ruptura
        BLE   ENDW03          While Núm.-Punto-Rupt.>0
        DECA                  Decrementa Núm.-Punto-Rupt.
        BSR   BP02            Establecer-Punto-Ruptura
        BRA   WHIL03          Siguiente-Punto-Ruptura
ENDW03  LDA   #1              Poner a 1 Siguiente-Punto-
        STA   NEXTBP,PCR      Ruptura
        STD   1,S
        RTS
```

La orden M

```
PROMPT  FCB   '>
SPACE   FCB   32              Espacio, en código ASCII
CMDM    PSHS  A,B,X           Guarda regs. empleados
        LDA   PROMPT,PCR
        BSR   OUTCH           Visualiza el aviso
        BSR   GETADD          Toma Posición-Actual
        TFR   D,X             La lleva a X
REPT01  LDB   ,X              Toma Valor-Actual
        BSR   PUTHEX          Lo visualiza
        LDA   SPACE,PCR
        BSR   OUTCH           Visualiza un Espacio
        BSR   GETVAL          Toma Nuevo-Valor
IF03    TSTA                  Si Valor-Nuevo no es un punto
        BLT   UNTL01
        BGT   ENDF03          Si no es un Return
        STB   ,X              Almacena Valor-Nuevo en
                              Posición-Actual
ENDF03  LEAX  1,X             Incrementa Posición-Actual
        TFR   X,D             Visualiza Posición-Actual
        BSR   DSPADD
        BRA   REPT
UNTL01  PULS  A,B,X,PC
```

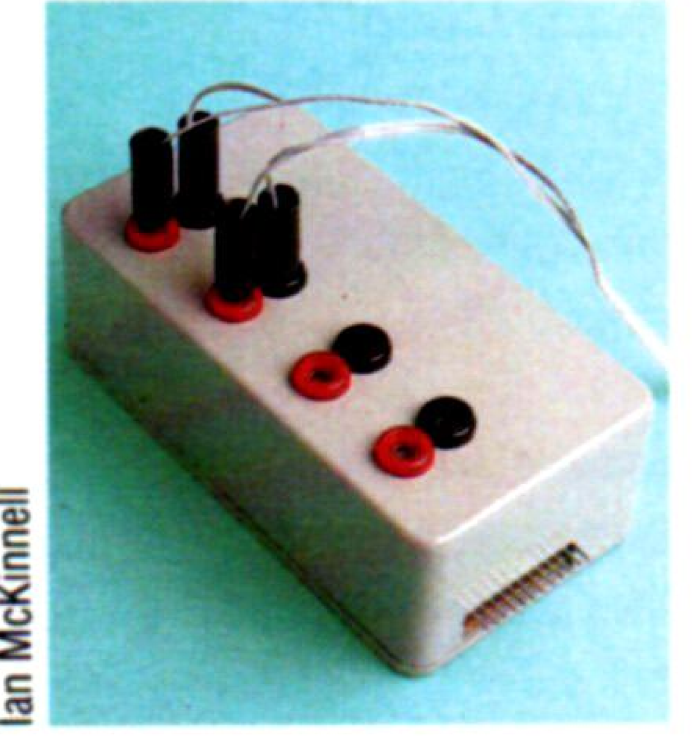
La suma de las partes

He aquí un repaso de las materias que hemos estudiado hasta el momento en este apartado

Los micros BBC y Commodore 64 poseen una disposición de entrada/salida similar que permite la comunicación con el mundo exterior a través de una puerta para el usuario, que consiste esencialmente en ocho patillas de datos y una conexión a tierra. Cada una de estas ocho patillas de datos está asociada a una posición particular de la memoria, denominada *registro de datos*, correspondiendo cada patilla a un bit en el registro. Una segunda posición, el registro de dirección de datos (RDD), controla la dirección de los datos que fluyen a o desde cada patilla. Si cualquiera de las patillas se establece en salida (bit RDD=1), en la patilla se induce un voltaje de + 5 V cada vez que el bit correspondiente se establece alto (en uno). Si el bit de datos se establece bajo, en la patilla se induce un voltaje de 0 V. Si bien la corriente que se suministra desde las patillas de datos de la puerta para el usuario no puede activar directamente dispositivos externos, sí se la puede utilizar para disparar un sistema de relé que permita encender o apagar voltajes mayores y/o sistemas que estén conectados a la red eléctrica.

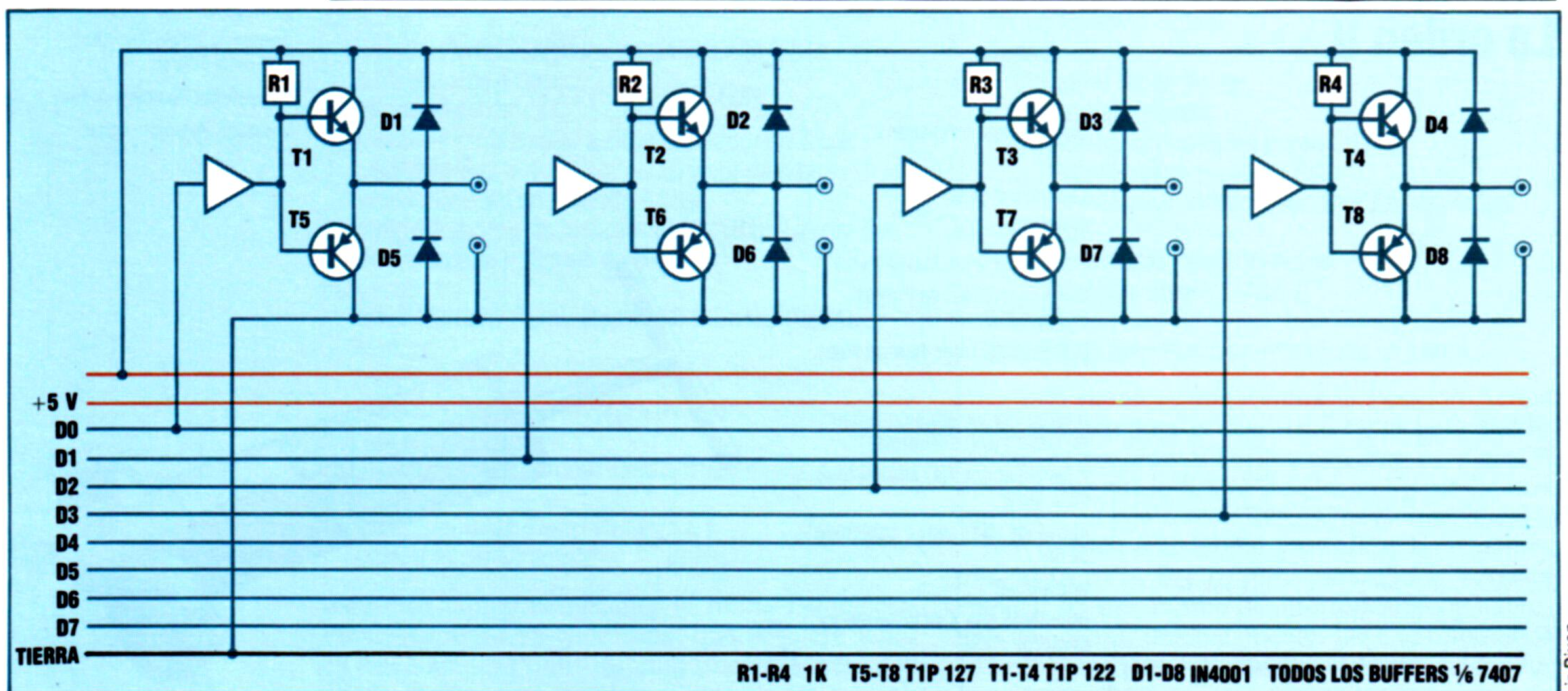
Cuando se establece una patilla para entrada (bit RDD=0), entonces el método de operación es bastante diferente. En este caso, el bit correspondiente del registro de datos se mantiene alto, bajando solamente si la patilla se conecta a tierra. Esto se puede emplear para controlar eventos del mundo exterior, conectando un lado de un sencillo interruptor a una patilla de datos y el otro lado a la toma de tierra de la puerta para el usuario. Cuando se mueve el interruptor, la patilla de datos conecta a tierra y el bit correspondiente del registro de datos sufre una transición de alto a bajo.

Las ocho líneas de datos y la toma de tierra se deben conectar de alguna forma a cada dispositivo del sistema de la puerta para el usuario y, de este modo, todo el sistema completo se designa alrededor de un bus común de nueve líneas, conectado cada dispositivo en la línea adecuada para su función particular. Este bus común se conecta a cada dispositivo mediante un conector minicon de 12 vías. Conectando a un conector macho del lado "interior" y un conector hembra del lado "exterior" en cada dispositivo, podemos realizar una "cadena margarita" de cualquier combinación de componentes del sistema entre sí.

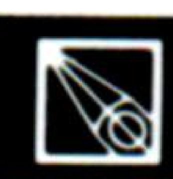

Ian McKinnell

La caja de salida

La caja de salida de bajo voltaje se conecta con el bus del sistema a través de un conector macho minicon de 12 vías, que se enchufa en su equivalente hembra de la caja buffer. La corriente suministrada por una patilla de datos activada es del orden de unos pocos miliamperios, insuficiente para activar un dispositivo como, por ejemplo, un motor eléctrico, pero suficiente para actuar a modo de corriente de conmutación a través de un transistor. Las líneas de datos de la 0 a la 3 las utiliza la caja de bajo voltaje. La activación (estado "alto") de una de estas líneas hace que el voltaje del transformador se conmute a través de un transistor al correspondiente conector de red de esta caja. Por lo tanto, con el voltaje de entrada se puede abastecer simultáneamente a cuatro dispositivos.



Liz Dixon



PUERTA PARA EL USUARIO

AMPLIACIÓN DEL SISTEMA

ENTRADAS

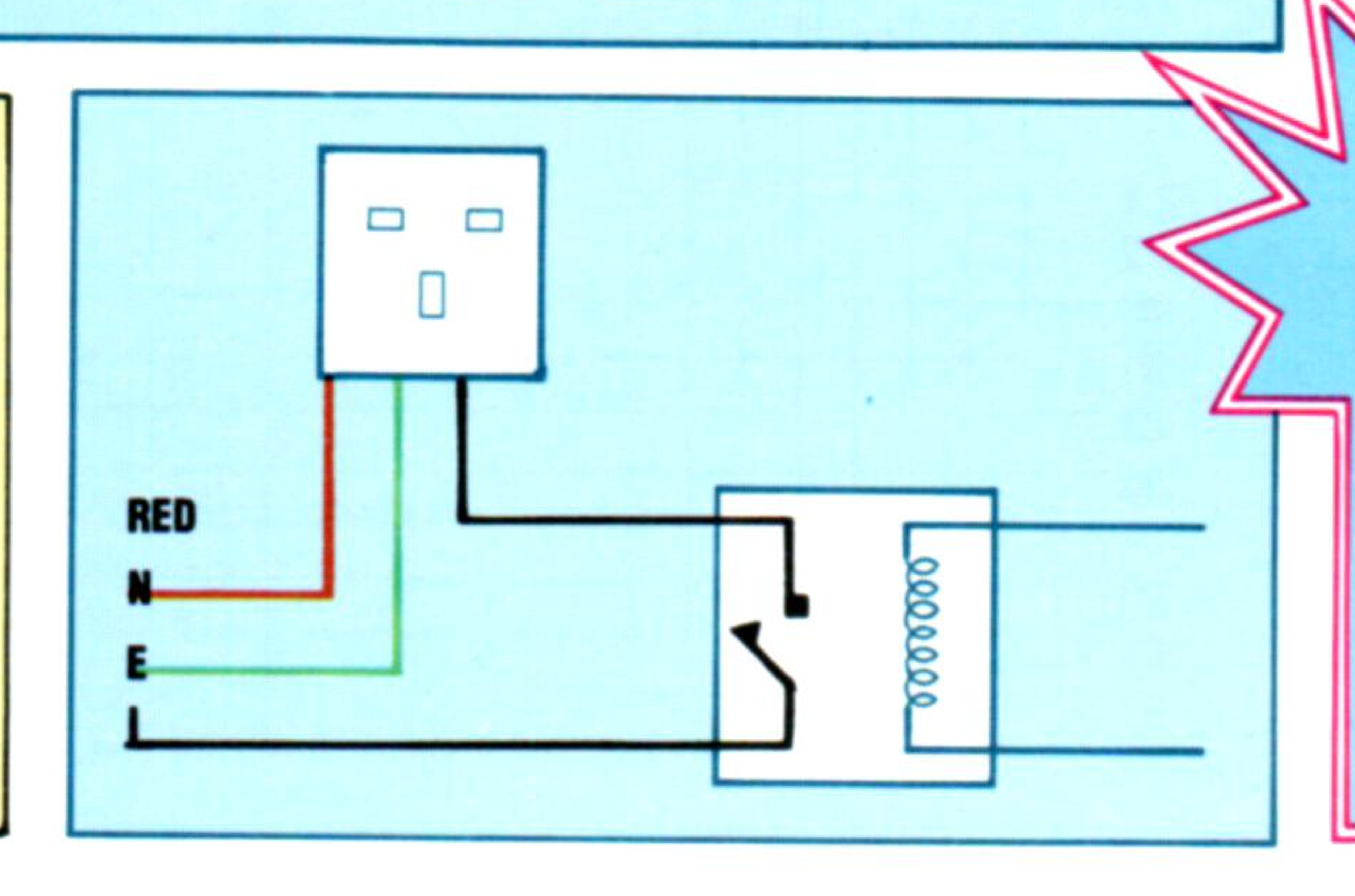
R1-8 240 Ω
R9-16 4,7 K
TODOS LOS BUFFERS 1/6 7407

La caja buffer

Es el dispositivo más importante del sistema de la puerta para el usuario. El sistema de circuitos protege los chips de E/S del ordenador contra cualquier intento de "sorber" demasiada corriente de una patilla de datos o por aplicar un voltaje de entrada peligroso. Además, acepta y regula una entrada de CD a CA desde un transformador, en la escala de 5 a 21 V. Esta entrada de voltaje se agrega al bus del sistema como un par de líneas extras, para utilizar con otros componentes del sistema. Con la caja buffer conectada se pueden efectuar entradas en la puerta para el usuario; los ocho conectores rojos corresponden a las ocho líneas de datos, los conectores negros proporcionan una toma a tierra separada para cada línea de datos. Hay montada en la caja una serie de ocho LED. Cada LED se ilumina si su línea de datos se desactiva

El relé de red

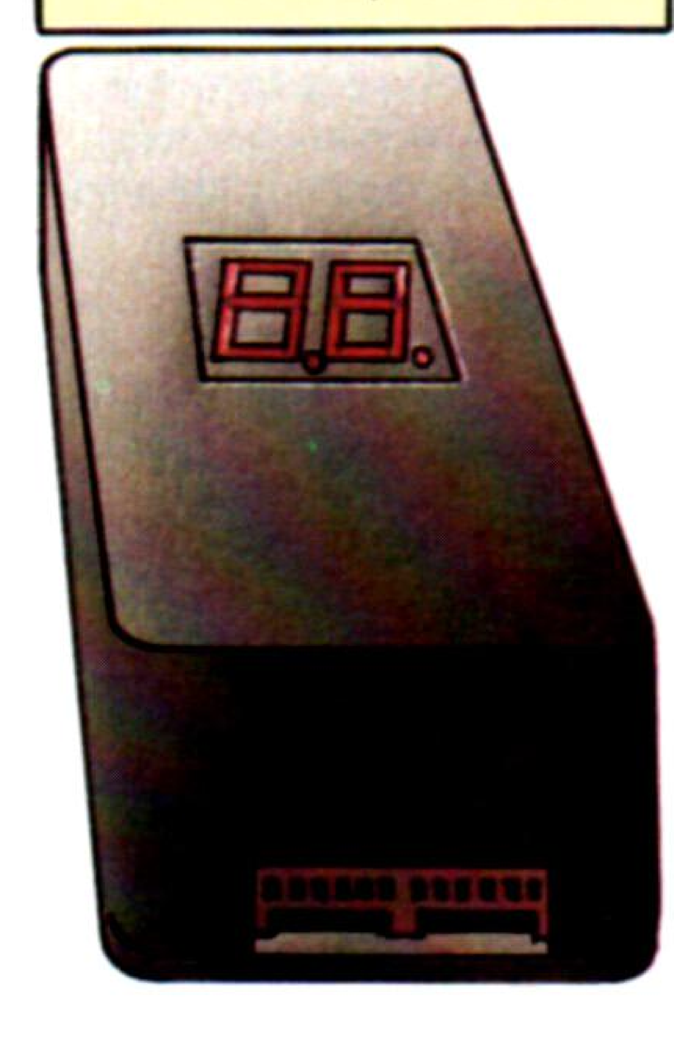
Se puede hacer que el voltaje del transformador conmute un voltaje de la red mediante un relé. Esta unidad se enchufa a la red y a una de las cuatro líneas de la caja de salida. Al establecer alto uno de los bits del registro de datos, se conmuta la alimentación del transformador al conector de salida correspondiente, que a su vez conmuta la alimentación de la red al conector de tres vías. Así es posible controlar aparatos eléctricos desde el ordenador. (Véase p. 1126.)

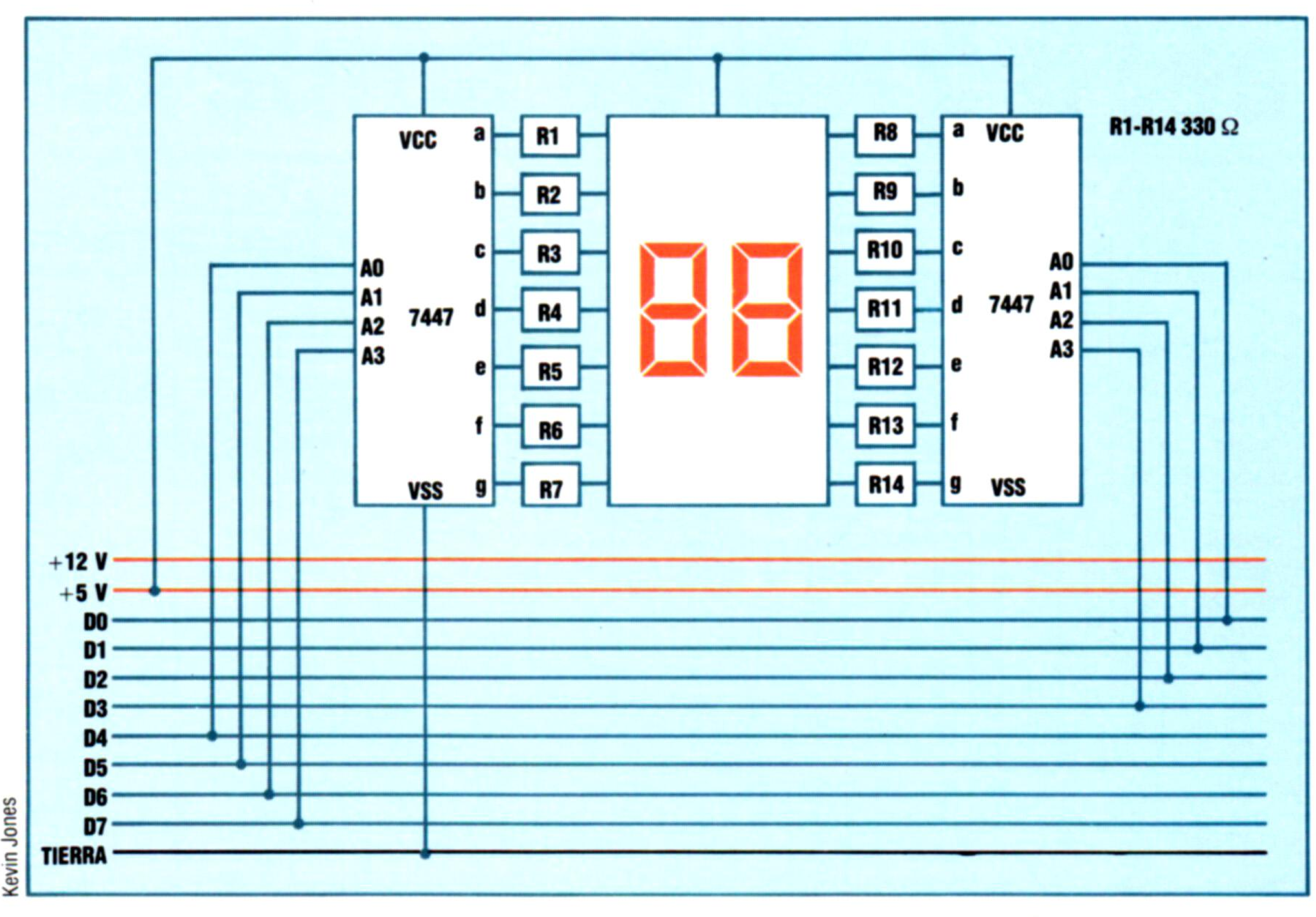


¡ATENCIÓN!
Todo lo que implica potencia eléctrica exige sumo cuidado.
- **Antes de comenzar a trabajar desconecte las fuentes de potencia.**
- **Compruebe conexiones y aislamientos mediante un tester.**
- **Evite los cortocircuitos.**

Visualización en siete segmentos

Requiere 4 entradas lógicas para visualizar los 16 dígitos hexa, de modo que utilizando las ocho líneas de entrada disponibles en el bus del sistema podemos activar dos de tales visualizaciones. Esta unidad visualizará, por consiguiente, el contenido del registro de datos de la puerta para el usuario como un par de dígitos hexa, y se puede conectar ya sea a la caja buffer o bien a un conector minicon hembra de la caja de salida

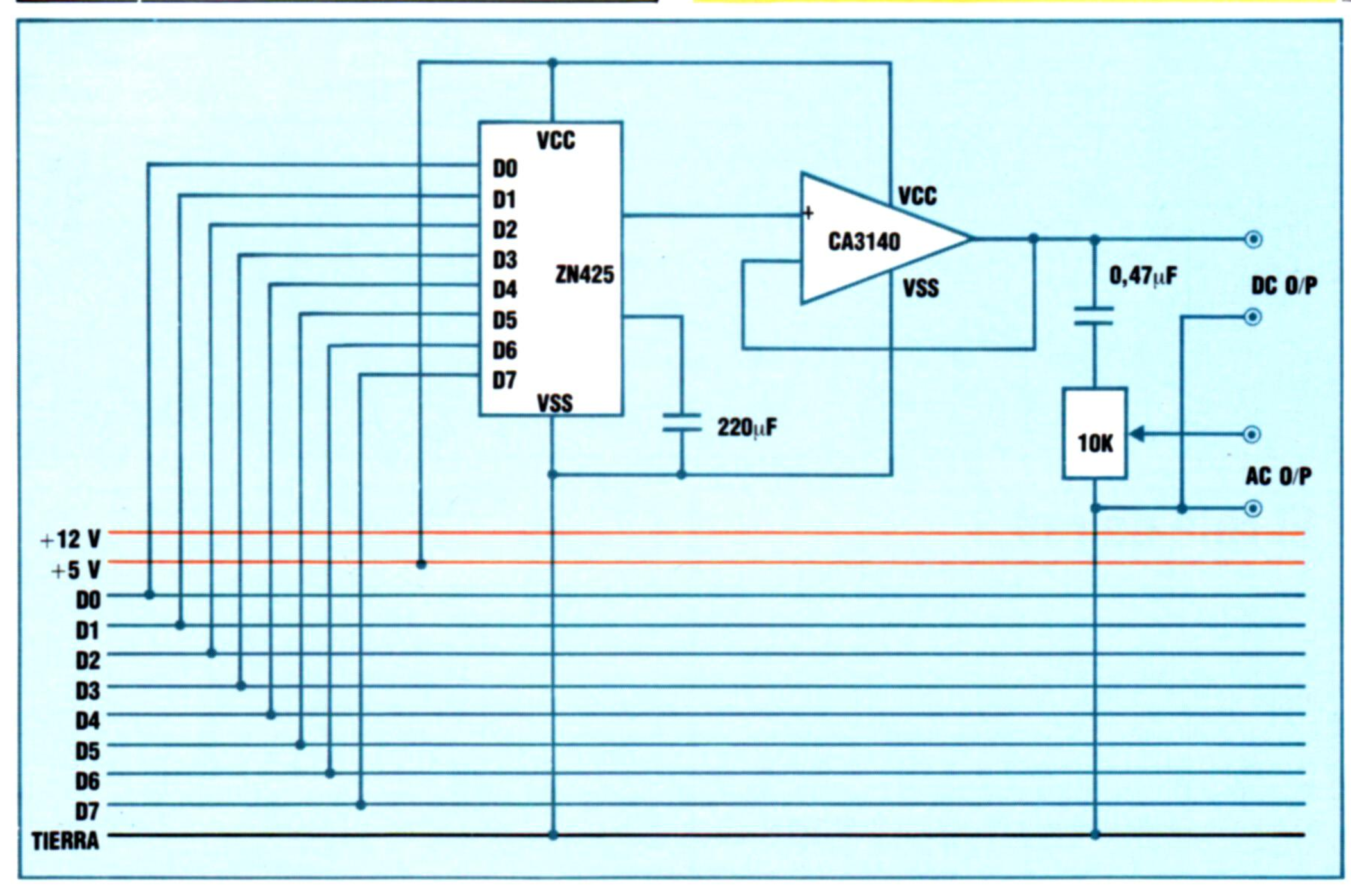




El convertidor D/A

El convertidor de digital a analógico convierte un valor del registro de datos entre 0 y 255 en un voltaje. La salida de la caja es de alrededor de 600 W, no lo suficientemente fuerte como para activar directamente motores u otros dispositivos de gran consumo, pero se la puede amplificar para hacerlo. Por otra parte, el convertidor se puede utilizar para generar sonido, ya sea a través de auriculares o bien de un amplificador de audio





Editorial Delta, S.A.



10064

9 788485 822836